केंद्रीय विद्यालय संगठन (मुंबई संभाग)

STUDY / SUPPORT MATERIAL

(Reviewed/Revised)

2013 – 14

कक्षा / CLASS – XI

अभिकलित्र - विज्ञान

COMPUTER SCIENCE (083)

KENDRIYA VIDYALAYA SANGTHAN

MUMBAI REGION

ACKNOWLEDGEMENTS

CHIEF PATRON : **SHRI AVINASH DIKSHIT**
COMMISSIONER,
KENDRIYA VIDYALAYA SANGATHAN
NEW DELHI

PATRON : **MS. R. KALAVATHI**
DEPUTY COMMISSIONER
KENDRIYA VIDYALAYA SANGATHAN
BHUBANESWAR REGION

ADVISORS : **SHRI M.V.R.K. SASTRY**
ASSISTANT COMMISSIONER
KENDRIYA VIDYALAYA SANGATHAN
BHUBANESWAR REGION

SHRI G.S.N. REDDY
ASSISTANT COMMISSIONER
KENDRIYA VIDYALAYA SANGATHAN
BHUBANESWAR REGION

CO-ORDINATOR : **SHRI S.K. CHOPDAR**
ASSISTANT COMMISSIONER
KENDRIYA VIDYALAYA SANGATHAN
BHUBANESWAR REGION

STUDY MATERIAL PREPARATION COMMITTEE

CONVENER : **SHRI S.K. BEHURA**
PRINCIPAL,
KENDRIYA VIDYALAYA NO. 1,
UNIT – IX, BHUBANESWAR

MEMBERS : **SH. ARUN KUMAR**
PGT(COMP.SC.),
KV NO.1, 2ND SHIFT, BHUBANESWAR

SH. KAMAL KANT GUPTA
PGT(COMP.SC.),
K.V. NO.-2, SV NAGAR ,VSKP

SH. ANIL KUMAR S S
PGT(COMP.SC.),
K.V. NO.-1, SV NAGAR,VSKP

SH. MUNENDRA PRAKASH
PGT(COMP.SC.),
K.V., WALTAIR,VSKP

PREFACE

Kendriya Vidyalayas are pace setting institutions spread across the length and breadth of this vast country including a few in the foreign lands. Text books are like the sumptuous meals but a few students do need a digestive tablets to digest the food. The condensed versions of the text books is what the study material is.

For the academic year 2012-13, the task of preparing the study material has been entrusted with Kendriya Vidyalaya Sangathan, BhubaneswarRegion for the students of Class XI in Computer Science.

Every effort has been made by four of your chosen PGTs (Computer Science) to present the material in brief but not compromising on the matter to the presented. Stress has equally been laid on the HOTs (Higher Older Thinking Skills) questions, as well as the simpler concepts. All the four PGTs (Computer Science) Sh. Arun Kumar, Sh. Kamal Kant Gupta, Sh. Anil Kumar S S, Sh. Munendra Prakash deserve the appreciation for their efforts.

I am sure, the support material will definitely of help to our students to prepare for the examinations besides helping them in understanding the subject.

(R. KALAVATHI)
DEPUTY COMMISSIONER
KVS, BHUBANESWAR REGION

REVIEW REPORT

I have the privilege in reviewing the support / study material for Computer Science (083) for class XI session 2013-14 and I am really thankful to our Esteemed Deputy Commissioner, KVS RO Mumbai, **Ms L CHARI,** for entrusting me this responsibility.

The study / support material of Computer Science (083) for class XI has been thoroughly reviewed in accordance with the curriculum and marking scheme of CBSE. Further, Unit I - **Computer Fundamental** and Unit III - **Programming Methodology** have been modified as per CBSE norms as these two units are having maximum theoretical portion. Also the CBSE has modified the marks distribution for Unit III and Unit IV, same is incorporated herein.

I am glad to express my sincere gratitude to **Ms R. KALAVATHI,** Deputy Commissioner, KVS RO Bhubaneswar, and the other team members for preparing such an exhaustive study material.

At the end I would also extend my sincere thanks to **Ms SARITA NASWA**, Asst. Comm., KVS RO Mumbai and our Principal **Ms PRAMILA PAL** for having extended timely guidance and support.

KAMAL KISHOR SWAMI
PGT (Comp. Sc.)
KV BHANDUP, Mumbai

SUPPORT MATERIAL

COMPUTER SCIENCE
2013-14

CBSE Mark Distribution for different Units

Sl. No	Unit Name	Marks
1	UNIT 1 Computer Fundamental	10
2	UNIT 2 Introduction to C++	14
3	UNIT 3 Programming Methodology	12
4	UNIT 4 Programming in C++	34
	Total Marks	**70**

Weightage to different forms of questions

S. No.	Forms of Question	Marks for each question	No. of Questions	Total Marks
1	Very Short Answer Questions (VSA)	01	09	09
2	Short Answer Questions- Type 1 (SA1)	02	13	26
3	Short Answer Questions- Type II (SAII)	03	05	15
4	Long Answer Questions- (LA)	04	05	20
		Total	32	70

Difficulty Level of Questions

S. N.	Estimated Difficulty Level	Percentage of questions
1	Easy	15%
2	Average	70%
3	Difficult	15%

INDEX

S.No.	Topics	PAGE NO.
1	Unit 1 : Computer Fundamentals	09
2	Unit 2 : Introduction to C++	42
3	Unit 3 : Programming methodology	54
4	Unit 4: Programming in C++	64
5	Sample Papers	135

Detailed syllabus

UNIT 1: COMPUTER FUNDAMENTALS (18 Theory + 6 Practical) Periods

Evolution of computers; Basics of computer and its operation; Functional Components and their interconnections, concept of Booting. Classification of Computers.

Software concepts: Types of Software - System Software, Utility Software and Application Software

System Software: Operating System, Complier, Interpreter and Assembler

Operating System: Need for Operating System, Functions of Operating System (Processor Management, Memory Management, File Management and Device Management), Types of Operating System-interactive (GUI based), Time Sharing, Real Time and Distributed, Commonly used operating system: UNIX, LINUX, Windows, Solaris, BOSS (Bharat Operating System Solutions); Mobile OS - Android, Symbian.

Utility Software: Anti Virus, File Management tools, Compression tools and Disk Management tools (Disk Cleanup, Disk Defragmenter, Backup).

Open Source Concepts: Open Source Software, Freeware, Shareware, Proprietary Software.

Application Software: Office Tools - Word Processor, Presentation Tool, Spreadsheet Package, Database Management System; Domain Specific tools - School Management System, Inventory Management System, Payroll System, Financial Accounting, Hotel Management, Reservation System and Weather Forecasting System.

Number System: Binary, Octal, Decimal, Hexadecimal and conversion between two different number systems.

Internal Storage encoding of Characters: ASCII, ISCII (Indian scripts Standard Code for Information Interchange), and UNICODE (for multilingual computing)

Microprocessor: Basic concepts, Clock speed (MHz, GHz), 16 bit, 32 bit, 64 bit processors; 128 bir processors; Types - CISC Processors (Complex Instruction set computing), RISC Processors (Reduced Instruction set computing), and EPIC (Explicitly parallel Instruction computing).

Memory Concepts: Units: Byte, Kilo Byte, Mega Byte, Giga Byte, Tera Byte, Peta Byte, Exa Byte, Zetta Byte, Yotta Byte.

Primary Memory: Cache, RAM, ROM

Secondary Memory: Fixed and Removable storage - Hard Disk Drive, CD/DVD Drive, Pen Drive, Blue Ray Disk.

Input Output Ports/ Connections: Serial, Parallel and Universal Serial Bus, PS-2 port, Infrared port, Bluetooth, Firewire.

UNIT 2: INTRODUCTION TO C++ (44 Theory + 36 Practical) Periods

Getting Started: C++ character set, C++ Tokens (Identifiers, Keywords, Constants, Operators,), Structure of a C++ Program (include files, main function), Header files - iostream.h, iomanip.h, **cout, cin;** use of I/O operators

(<<and>>), Use of endl and setw (), Cascading of I/O operators, Error Messages; Use of editor, basic commands of editor, compilation, linking and execution.

Data Types, Variables and Constants: Concept of Data types; Built-in Data types: **char, int, float and double;** Constants: Integer Constants, Character constants - \n, \t, \b), Floating Point Constants, String Constants; Access modifier: const; Variables of built-in-data types, Declaration/Initialization of variables, Assignment statement, Type modifier: **signed, unsigned, long**

Operator and Expressions: Operators: Arithmetic operators (-,+,*,/,%), Unary operator (-), Increment (++) and Decrement (--) Operators, Relation operator (>,>=,<=,=,!=), Logical operators (!,&&,II), Conditional operator: <condition>?<if false>; Precedence of Operators; Automatic type conversion in expressions, Type casting; C++ shorthands (+=,- =,*=,/=,%=)

UNIT 3: PROGRAMMING METHODOLOGY (28 Theory + 10 Practical) Periods

General Concepts: Modular Approach, Clarity and Simplicity of Expressions, Use of proper names for Identifiers, Comments, Indentation; Documentation and Program Maintenance; Running and Debugging programs, Syntax Errors, Run-Time Errors, Logical Errors

Problem solving Methodologies: Understanding of the problem, solution for the problem, identifying minimum number of inputs required for output, writing code to optimizing execution time and memory storage, step by step solution for the problem, breaking down solution into simple steps (modular approach), identification of arithmetic and logical operations required for solution; Control Structure- conditional control and looping (finite and infinite).

Problem Solving: Introduction to Algorithms/Flowcharts.

UNIT 4: PROGRAMMING IN C++ (50 Theory + 48 Practical) Periods

Flow of control:

Conditional statements: if else, Nested **if, switch..case..default,** use of conditional operator, Nested **switch..case, break** statement (to be used in **switch..case only);** Loops: **while, do - while, for** and Nested loops

Inbuilt Functions

Header file Categorization Header File Function

Standard input/output functions stdio.h gets ( ), puts ( )

Character Functions Ctype.h isalnum ( ), isalpha ( ), isdigit ( ), islower ( ), isupper ( ), tolower ( ), toupper ( )

String Function string.h strucy ( ), strcat ( ), strcmpi ( ), strev ( ), strlen ( ), strupur ( ), strlwr ( ), strlen ( ), strcmp ( ),

Mathematical Functions math.h fabs ( ), pow ( ), sgrt ( ), sin ( ), cos ( ), abs ( )

Other Functions stdlib.h randomize ( ), random ( )

User Defined Functions :

Introduction to user-defined function and its requirements.

Defining a function; function prototype, Invoking/calling a function, passing arguments to function, specifying argument data types, default argument, constant argument, call by value, call by reference, returning values from a function, calling functions with arrays, scope rules of functions and variables local and global variables.

Structured Data Type:

Arrays: Introduction to Array and its advantages.

One Dimensional Array : Declaration/initialization of One-dimensional array, inputting array elements, accessing array elements, manipulation of array elements (sum of elements, product of elements, average of elements, linear search, finding maximum/minimum value) Declaration / Initialization of a String, string manipulations (counting vowels/ consonants/ digits/ special characters, case conversion, reversing a string, reversing each word of a string)

Two-dimensional Array

Declaration/initialization of a two-dimensional array, inputting array elements accessing array elements, manipulation of array elements (sum of row element, column elements, diagonal elements, finding maximum / minimum values)

User-defined Data Types: Introduction to user defined data types.

Structure

Defining a Structure (Keyword Structure), declaring structure variables, accessing structure elements, passing structure to functions as value and reference argument/parameter, function returning structure array of structure, passing an array of structure as an argument/ a parameter to a function. Defining a symbol name using **typedef** keyword and defining a macro using **#define** preprocessor directive.

Unit-1

Computer Fundamentals

After studying this session students will be able to:

- *Learn the evolution of computers*
- *Learn about various generations of computer*
- *Understand the basic operation of a computer*
- *Study the functional components and their interconnections*
- *Understand the concept of booting*
- *Learn about classification of computers*

Introduction

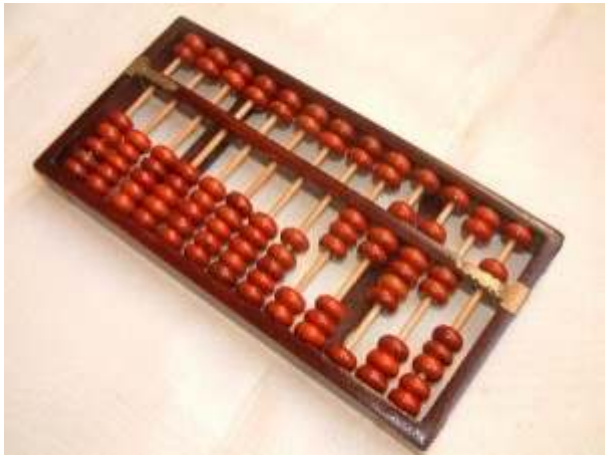
Computers are seen everywhere around us, in all spheres of life. May it be the field of education and research, travel and tourism, weather forecasting, social networking, ecommerce or any other, computers have now become an indispensable part of our lives. The manner, in which computers have revolutionised our lives because of their accuracy and speed of performing a job, is truly remarkable. Today no organization can function without a computer. In fact various organizations are trying to become paper free owing to benefits of computers. But the computers of today have evolved over the years from a simple calculating device to the portable high speed computers that we see today.

Evolution of Computers

The growth of computer industry started with the need for performing fast calculations. The manual method of computing was slow and prone to errors. So attempts were made to develop faster calculating devices. The journey that started from the first calculating device i.e. Abacus has led us today to extremely high speed calculating devices. Let us first have a look at some early calculating devices and then we will explore various generations of computer.

Abacus

Abacus was discovered by the Mesopotamians in around 3000 BC. An abacus consisted of beads on movable rods divided into two parts. (Fig-1) Addition and multiplication of numbers was done by using the place value of digits of the numbers and position of beads in an abacus.



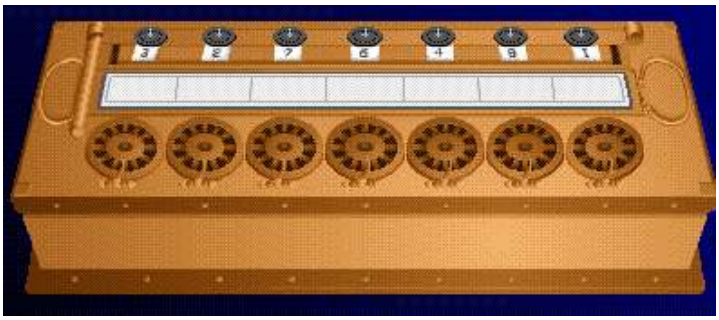
The Chinese further improved on the abacus so that calculations could be done more easily. Even today abacus is considered as an apt tool for young children to do calculations. In an abacus, each row is thought of as a ten's place. From right to left, row no-1 represents the one's column and the second column represents ten's place. The third column represents the hundred's place and so on. The starting position of the top beads (representing the value of five) is always towards the top wall of the abacus while the lower beads (representing the value of one) will always be pushed towards the lower wall as a starting position.

Napier's Logs and Bones

The idea of logarithm was developed by John Napier in 1617. He devised a set of numbering rods known as Napier's bones through which both multiplication and division could be performed. These were numbered rods which could perform multiplication of any number by a number in the range of 2-9. There are 10 bones corresponding to the digits 0-9 and there is also a special eleventh bone that is used to represent the multiplier. By placing bones corresponding to the multiplier on the left side and the bones corresponding to the digits of the multiplicand on the right , the product of two numbers can be easily obtained.

Pascaline

Blaise Pascal, a French mathematician invented an adding machine in 1642 that was made up of gears and was used for adding numbers quickly. This machine was also called Pascaline and was capable of addition and subtraction along with carry-transfer capability. It worked on clock work mechanism principle. It consisted of various numbered toothed wheels having unique position values. The addition and subtraction operations was performed by controlled rotation of these wheels.

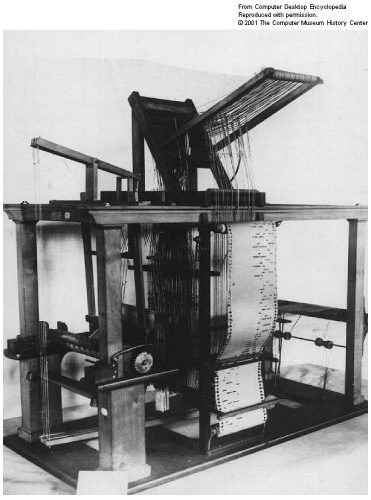


Leibnitz's Calculator

In 1673 Gottfried Leibnitz, a German mathematician extended the capabilities of the adding machine invented by Pascal to perform multiplication and division as well. The multiplication was done through repeated addition of numbers using a stepped cylinder each with nine teeth of varying lengths.

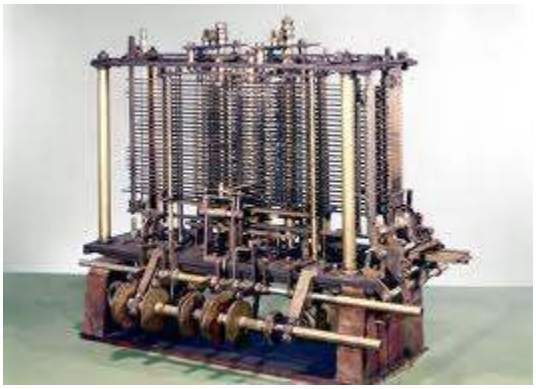
Jacquard's Loom

In order to make the cotton weaving process automatic, Joseph Jaquard devised punch cards and used them to control looms in 1801. The entire operation was under a program's control. Through this historic invention, the concept of storing and retrieving information started.



Difference engine and Analytical Engine

Charles Babbage, an English mathematician developed a machine called Difference Engine in 1822 which could calculate various mathematical functions, do polynomial evaluation by finite difference and theoretically could also solve differential equations. Thereafter in 1833, he designed the Analytical Engine which later on proved to be the basis of modern computer. This machine could perform all the four arithmetic operations as well as comparison. It included the concept of central processor, memory storage and input-output devices. Even the stored information could be modified. Although the analytical engine was never built that time but Babbage established the basic principles on which today's modern computers work. Both these great inventions earned him the title of **'Father of Modern Computers'.**



In 1944 Prof Howard Aiken in collaboration with IBM constructed an electromechanical computer named **Mark 1** which could multiply two 10 digit numbers in 5 seconds. This machine was based on the concept of Babbage's Analytical engine and was the first operational general purpose computer which could execute preprogrammed instructions automatically without any human intervention.

In 1945, Dr. John Von Neumann proposed the concept of a stored program computer. As per this concept the program and data could be stored in the same memory unit. According to Von Neumann architecture, the processor executes instructions stored in the memory of the computer. Since there is only one communication channel, the processor at a time can either fetch data or an instruction. That means at one point of time either the data or an instruction can be picked (fetched) from the storage unit for execution by the processor. Hence execution takes place in sequential manner. This limitation of Von Neumann Computer is known as Von Neumann bottleneck. EDVAC (**E**lectronic **D**iscrete **V**ariable **A**utomatic **C**omputer) was the first stored program computer developed in 1952. After the invention of first electronic computer ENIAC (**E**lectronic **N**umerical **I**ntegrator and **C**alculator) in 1946, the computer technology improved tremendously and at very fast pace.

Generation of Computers

Growth in the computer industry is determined by the development in technology. Each phase/generation of computer development is characterized by one or more hardware/software developments that distinctly improved the performance of the computers of that generation. Based on various stages of development, computers can be divided into different generations.

The First Generation (1942-1955)

The first generation computers used the concept of 'stored program' and were characterized by vacuum tubes. A vacuum tube is a delicate glass device that can control and amplify electronic signals. The first generation computers were made using thousands of vacuum tubes and were the fastest calculating devices of their time. These computers were very large in size, consumed lot of electricity and generated lot of heat. UNIVAC 1 was the first electronic computer of this generation and was used for business applications.

Examples: ENIAC , EDVAC , UNIVAC 1

The Second Generation (1955–1964)

The second generation computers were characterized by transistors. A transistor is a solid state semiconductor device that revolutionized the electronic industry. Transistors were smaller, highly reliable, consumed less electricity and generated less heat. Also magnetic core memories were developed during this generation. These are tiny ferrite rings that can be magnetized in either clockwise or anticlockwise direction so as to represent binary 1 or binary 0. Magnetic cores were used as primary memories. Later magnetic disks also came into existence and were used as secondary storage devices.
All these new developments – transistors, magnetic core memory and magnetic disk storage devices made the computers more powerful and reliable. This further led to the existence of operating systems. Programming languages like FORTRAN, COBOL, Algol etc. also developed. Commercial applications of the computer increased and now the computers were used in business and industries for applications like payroll, employee management, inventory control etc. IBM 1401 and IBM 1620 were popular computers of this generation.
Examples: IBM 1401, IBM 1620, UNIVAC 1108

The Third Generation (1964-1975)

In 1964, the Integrated Circuits or ICs or chips revolutionized the electronic industry and started the third generation of computers. An IC is a small silicon chip or wafer made up of extremely purified silicon crystals. It has numerous transistors, capacitors, resistors and other elements of an electronic circuit. A small scale integration (SSI) chip used to have about 10 transistors on a single chip and a medium scale integration (MSI) chip had about 100 transistors per chip. The size of memories also increased. Various mainframe computers and minicomputers were developed during this generation. Even operating systems with multitasking and multiprogramming features (you will learn about these terms in the next chapter) were developed. Since ICs made the computers highly reliable, relatively inexpensive and faster, computers these days were found in areas of education, small businesses and offices along with industrial and business applications. IBM 360 was a very popular third generation computer.
Examples: IBM 360 series, Honeywell 6000 series

The Fourth Generation (1975 onwards)

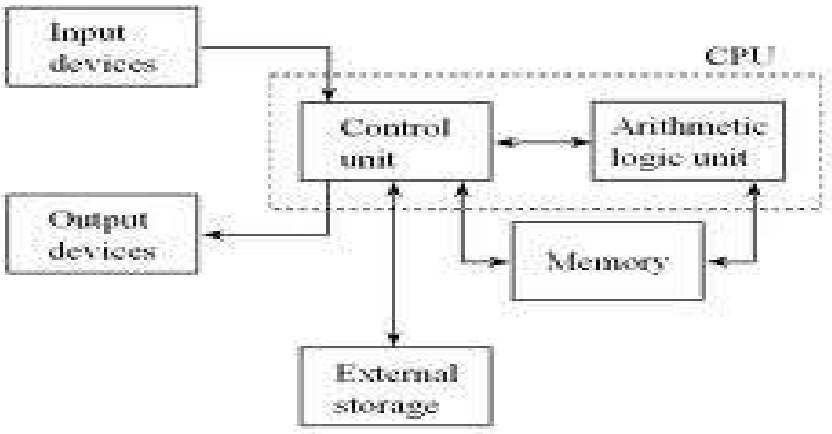
In this generation Large Scale Integration (LSI) and Very Large scale integration (VLSI) technology was used by which up to 300,000 transistors were used on a single chip. Thus integration of complete CPU on a single chip was achieved in 1971 and was named microprocessor which marked the fourth generation of computers. The computers based on microprocessor technology had faster accessing and processing speeds. In addition to this the increased memory capacity further made the computers more powerful and also more efficient operating systems were developed for these computers. New concepts of microprogramming, application software, databases, virtual memory etc were developed and used. The computers that we use today belong to this generation. These portable computers can be carried from one place to another owing to their compact size. They are much more accurate. Even memory sizes have become phenomenal. Commercial production of these computers is easier and they are the least expensive, compared to the earlier generation computers. Also computer networks starting coming up during this generation. It is today one of the most popular means to interact and communicate with people.

Fifth Generation Computers

Fifth Generation computers are still under development. This generation is based on the concept of artificial intelligence. In simple terms the computers of this generation are supposed to behave like humans. The principles of parallel processing (many processors are grouped together) and

superconductivity are being used to develop devices that respond to human languages and will have the ability to apply previously gained knowledge to execute a task. They will let them make decisions of their own to execute a task. Some applications like voice recognition, visual recognition are a step in this very direction.

ARCHITECTURE OF COMPUTER



Input Devices: Those devices which help to enter data into computer system. Eg. Keyboad, Mouse, Touchscreen, Barcode Reader, Scanner, MICR, OMR etc.

Bar code Reader

MICR used in Bank

OMR(Used for answer sheet evaluation)

Output Devices: Those devices which help to display the processed information. Eg. Monitor, Printer, Plotter, Projector

Printer

Plotter

Projector

Central Processing Unit (CPU)

Once the information is entered into the computer by the input device, the processor processes it. The CPU is called the brain of the computer because it is the control centre of the computer. As the CPU is located on a small chip, it is also called the microprocessor. It first fetches instructions from memory and then interprets them so as to know what is to be done. If required, data is fetched from memory or input device. Thereafter CPU executes or performs the required computation and then either stores the output or displays on the output device. The CPU has three main components which are responsible for different functions – Arithmetic Logic Unit (ALU) , Control Unit (CU) and Memory registers.

Arithmetic and Logic Unit (ALU)

The ALU, as its name suggests performs mathematical calculations and takes logical decisions. Arithmetic calculations include addition, subtraction, multiplication and division. Logical decisions involve comparison of two data items to see which one is larger or smaller or equal.

Control Unit

The Control unit coordinates and controls the data flow in and out of CPU and alsocontrols all the operations of ALU, memory registers and also input/output units. It is also responsible for carrying out all the instructions stored in the program. It decodes the fetched instruction, interprets (understands) it and sends control signals to input/output devices until the required operation is done properly by ALU and memory.

Memory Registers

A register is a temporary unit of memory in the CPU. These receive data/information and then this data/information is held in them as per the requirement. Registers can be of different sizes(16 bit , 32 bit , 64 bit and so on) and each register inside the CPU has a specific function like storing data, storing an instruction, storing address of a location in memory etc. The user registers can be used by an assembly language programmer for storing operands, intermediate results etc. Accumulator (ACC) is the main register in the ALU and contains one of the operands of an operation to be performed in the ALU.

Memory: It facilitates the remembrance power to computer system. It refers to the physical devices used to store programs (sequences of instructions) or data (e.g. program state information) on a temporary or permanent basis for use in a computer or other digital electronic device. The term primary memory is used for the information in physical systems which are fast (i.e. RAM), as a distinction from secondary memory, which are physical devices for program and data storage which are slow to access but offer higher memory capacity. Primary memory stored on secondary memory is called virtual memory. Primary Memory can be categorized as Volatile Memory & Non-Volatile Memory.

Volatile memory RAM)

Volatile memory is computer memory that requires power to maintain the stored information. Most modern semiconductor volatile memory is either Static RAM or dynamic RAM.

SRAM retains its contents as long as the power is connected and is easy to interface to but uses six transistors per bit.

Dynamic RAM is more complicated to interface to and control and needs regular refresh cycles to prevent its contents being lost.

However, DRAM uses only one transistor and a capacitor per bit, allowing it to reach much higher densities and, with more bits on a memory chip, be much cheaper per bit. SRAM is not worthwhile for desktop system memory, where DRAM dominates, but is used for their cache memories..

Non Volatile Memory (ROM)

Non-volatile memory is computer memory that can retain the stored information even when not powered.

Examples of non-volatile memory are flash memory and ROM/PROM/EPROM/EEPROM memory (used for firmware such as boot programs).

Cache Memory:

Cache memory is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. As the microprocessor processes data, it looks first in the cache memory and if it finds the data there (from a previous reading of data), it does not have to do the more time-consuming reading of data from larger memory. It is of two types- L1 cache is on the same chip as the microprocessor. L2 is usually a separate static RAM (SRAM) chip.

Secondary Memory:

A. Hard Disk (Local Disk)
B. Optical Disks: CD-R, CD-RW, DVD-R, DVD-RW
C. Pen Drive
D. Floppy Disks
F. Memory Cards
G. External Hard Disk
H. Blu Ray Disk

Blu-Ray Disk:

Blu-ray (not Blue-ray) also known as Blu-ray Disc (BD), is the name of a new optical disc format. The format offers more than five times the storage capacity of traditional DVDs and can hold up to 25GB on a single-layer disc and 50GB on a dual-layer disc. While current optical disc technologies such as DVD, DVD±R, DVD±RW, and DVD-RAM rely on a red laser to read and write data, the new format uses a blue-violet laser instead, hence the name Blu-ray.

Units of Memory:

The smallest unit is bit, which mean either 0 or 1.

1 bit = 0 or 1
1 Byte = 8 bit
1 Nibble = 4 bit
1 Kilo Byte = 1024 Byte= 2^{10} Byte
1 Mega Byte = 1024 KB= 2^{10} KB
1 Gega Byte = 1024 MB= 2^{10} MB
1 Tera Byte = 1024 GB= 2^{10} GB
1 Peta Byte =1024 TB= 2^{10} TB
1 Exa Byte =1024 PB= 2^{10} PB
1 Zetta Byte = 1024 EB= 2^{10} EB
1 Yotta Byte = 1024 ZB= 2^{10} ZB

Booting

The process of loading the system files of the operating system from the disk into the computer memory to complete the circuitry requirement of the computer system is called booting.

Types of Booting:

There are two types of booting:

- **Cold Booting:** If the computer is in off state and we boot the computer by pressing the power switch 'ON' from the CPU box then it is called as cold booting.

- **Warm Booting:** If the computer is already 'ON' and we restart it by pressing the 'RESET' button from the CPU box or CTRL, ALT and DEL key simultaneously from the keyboard then it is called warm booting.

Types of Computer

On the basis of working principle

a) Analog Computer

An analog computer is a form of computer that uses *continuous* physical phenomena such as electrical, mechanical, or hydraulic quantities to model the problem being solved.

Eg: Thermometer, Speedometer, Petrol pump indicator, Multimeter

b) Digital Computer

A computer that performs calculations and logical operations with quantities represented as digits, usually in the binary number system.

c) Hybrid Computer (Analog + Digital)

A combination of computers those are capable of inputting and outputting in both digital and analog signals. A hybrid computer system setup offers a cost effective method of performing complex simulations. The instruments used in medical science lies in this category.

On the basis of Size

a) Super Computer

The fastest type of computer. Supercomputers are very expensive and are employed for specialized applications that require immense amounts of mathematical calculations. For example, weather forecasting requires a supercomputer. Other uses of supercomputers include animated graphics, fluid dynamic calculations, nuclear energy research, and petroleum exploration. PARAM, Pace & Flosolver are the supercomputer made in india.

b) Mainframe Computer

A very large and expensive computer capable of supporting hundreds, or even thousands, of users simultaneously. In the hierarchy that starts with a simple microprocessor (in watches, for example) at the bottom and moves to supercomputers at the top, mainframes are just below supercomputers. In some ways, mainframes are more powerful than supercomputers because they support more simultaneous programs. But supercomputers can execute a single program faster than a mainframe.

c) Mini Computer

A midsized computer. In size and power, minicomputers lie between *workstations* and *mainframes*. In the past decade, the distinction between large minicomputers and small mainframes has blurred, however, as has the distinction between small minicomputers and workstations. But in

general, a minicomputer is a multiprocessing system capable of supporting from 4 to about 200 users simultaneously. Generally, servers are comes in this category.

d) Micro Computer

i. **Desktop Computer:** a personal or micro-mini computer sufficient to fit on a desk.

ii. **Laptop Computer:** a portable computer complete with an integrated screen and keyboard. It is generally smaller in size than a desktop computer and larger than a notebook computer.

iii. **Palmtop Computer/Digital Diary /Notebook /PDAs:** a hand-sized computer. Palmtops have no keyboard but the screen serves both as an input and output device.

e) Workstations

A terminal or desktop computer in a network. In this context, workstation is just a generic term for a user's machine (client machine) in contrast to a "server" or "mainframe."

Software Concepts

After studying this chapter the student will be able to:

- *Learn different types of Software*
- *System Software (Operating system, Language Processors)*
- *Utility Software (Antivirus, Compression tools, Backup, Disk Defragmentor)*
- *Application Software (General Purpose and Customized)*
- *Study the need, functions and types of operating system*
- *Study some commonly used operating systems- UNIX, LINUX, Windows, Solaris, BOSS*
- *Study mobile operating systems – Android and Symbian*
- *Understand Open Source Concepts - Open Source Software, Freeware, Shareware, Proprietary Software.*

Software can be broadly are categorized as

- System Software
- Application Software
- Utility Software

System Software

System Software is the software that is directly related to coordinating computer operations and performs tasks associated with controlling and utilizing computer hardware. These programs assist in running application programs and are designed to control the operation of a computer system. System software directs the computer what to do, when to do and how to do. System software can be further categorized into

- Operating System
- Language Translators

Operating System

An Operating system is the most important system software. It is a set of programs that control and supervise the hardware of a computer and also provide services to application software, programmers and users. It manages all hardware and software, input, output and processing activities within the computer system, the flow of information to and from the processor, sets priorities for handling different tasks, and so on. Without operating system a computer cannot do anything useful. When a computer is switched on, the operating system is the first program that is loaded onto its memory. A user cannot communicate directly with the computer hardware, so the operating system acts as an interface between the user and the computer hardware. Some of the popular operating systems used in personal computers are DOS, Windows, Unix, Linux, Solaris, etc.

An operating system can be a Single User or a Multiuser operating system. A single user operating system allows only one user to work at any time but a multiuser operating system allows two or more users to use a powerful computer at the same time. For example Windows 7 is a single user operating system while Linux is a multiuser operating system.

Need for an Operating System

Operating system provides a platform, on top of which, other programs, called application programs can run. As discussed before, it acts as an interface between the computer and the user. It is designed in such a manner that it operates, controls and executes various applications on the computer. It also allows the computer to manage its own resources such as memory, monitor, keyboard, printer etc.

Our choice of operating system, therefore, depends to a great extent on the CPU and the other attached devices and the applications we want to run. The operating system controls the various system hardware and software resources and allocates them to the users or programs as per their requirement.

Functions of an Operating System

An operating system has variety of functions to perform. Some of the prominent functions of an operating system can be broadly outlined as:

✡ **Processor Management:** This deals with management of the Central Processing Unit (CPU). The operating system takes care of the allotment of CPU time to different processes. This is called **scheduling**.

Two types of scheduling techniques are employed by an operating system :

- **Priority Scheduling:** Each task is given CPU time according to the priority assigned to that task. The program with higher priority will be given CPU time before a program with lower priority. The CPU executes the task till it is completed or there is some interrupt request i.e. till the time operating system has to stop (interrupt) the current task due to an unavoidable job request. The major drawback of Priority scheduling is that even a small job has to wait for a long time when a long duration job with higher priority is being executed.
- **Round Robin Scheduling:** This type of scheduling technique is also known as Time Sharing Scheduling. In this, each program or task is given a fixed amount of time to execute. The CPU continues with the execution till either the allotted time is over or there is some interrupt request or the task is completed before the allotted time. If the task is not completed at the end of the allotted time, it is put at the end of the queue. So each task gets its allotted share of CPU time. This scheduling technique improves the response time and provides an interactive environment. Hence time sharing operating system is very useful in network environment as each user is allowed to share the network resources.

✡ **Device Management:** The Operating System communicates with hardware and the attached devices and maintains a balance between them and the CPU. This is all the more important because the CPU processing speed is much higher than that of I/O devices. In order to optimize the CPU time, the operating system employs two techniques - Buffering and Spooling.

- **Buffering:** In this technique the temporary storage of input and output data is done in Input Buffer and Output Buffer. Once the signal for input or output is sent to or from the CPU respectively, the operating system through the device controller moves the data from the input device to the input buffer and for the output device to the output buffer. When the signal is sent to/from the operating system to the respective device controllers, the program doesn't wait rather it returns to its processing. In case of input, if the buffer is full, the operating system sends a signal to the program which processes the data stored in the buffer. When the buffer becomes empty, the program informs the operating system which reloads the buffer and the input operation continues. Similarly for output when the program being executed has to display some output, it fills the buffer and then informs the operating system. Thereafter the operating system empties the buffer by sending data to the output device and in the meantime the program fills another buffer. This technique is called **overlapped processing**. This is because while the operating system reloads one buffer, the executing program doesn't stop as it is able to retrieve/fill data from/in another buffer.
- **Spooling (Simultaneous Peripheral Operation on Line):** This is a device management technique used for processing of different tasks on the same input/output device. Say for example there are various users on a network sharing the same printer. At one point of time more than one user might give print command. The speed of the printer is very slow as compared to the CPU processing. So the operating system temporarily stores the data of every user on the hard disk of the computer to which

the printer is attached. The individual users need not wait for the printing process to be complete. Instead the operating system sends the data from to hard disk to the printer one by one.

✡ **Memory management:** In a computer, both the CPU and the I/O devices interact with the memory. When a program needs to be executed it is loaded onto the main memory till the execution is complete. Thereafter that memory space is freed and is available for other programs. The common memory management techniques used by the operating system are Partitioning and Virtual Memory.

- **Partitioning:** The total memory is divided into various partitions of same size or different sizes. This helps to accommodate number of programs in the memory. The partition can be fixed i.e. remains same for all the programs in the memory or variable i.e. memory is allocated when a program is loaded on to the memory. The later approach causes less wastage of memory but in due course of time, it may become fragmented.
- **Virtual Memory:** This is a technique used by the operating system by virtue of which the user can load the programs which are larger than the main memory of the computer. In this technique the program is executed even if the complete program is not loaded on to the main memory. The operating system divides the main memory into equal sizes called pages. A part of the program resides in the main memory and is called the **active set.** The rest is in the secondary storage device in the form of tracks/sectors or blocks. With the help of **Page Map Tables (PMT)**, the operating system keeps track which page of main memory is storing which block of secondary memory. A virtual address (which is not the real physical address) is mapped either to the main memory or the secondary memory. Hence virtual memory allows more programs and even larger programs to be executed in the main memory leading to efficient memory utilization.

✡ **File Management:** The operating System manages the files, folders and directory systems on a computer**.** Any data on a computer is stored in the form of files and the operating system keeps information about all of them using **File Allocation Table (FAT).** The FAT stores general information about files like filename, type (text or binary), size, starting address and access mode (sequential/indexed sequential/direct/relative). The file manager of the operating system helps to create, edit, copy, allocate memory to the files and also updates the FAT. The operating system also takes care that files are opened with proper access rights to read or edit them.

Types of Operating System

OS are classified into the following types depending on their capability of processing

✡ **Single User** and **Single Task OS:** It is used on a standalone single computer for performing a single task. Operating systems for Personal Computers (PC) are single-user OS. Single user OS are simple operating system designed to manage one task at a time. MS-DOS is an example of single user OS.

✡ **Multiuser OS** is used in mini computers or mainframes that allow same data and applications to be accessed by multiple users at the same time. The users can also communicate with each other. Linux and UNIX are examples of multiuser OS.

✡ **Multiprocessing OS** have two or more processors for a single running process. Processing takes place in parallel and is also called *parallel processing*. Each processor works on different parts of the same task, or, on two or more different tasks. Since execution takes place in parallel, they are used for high speed execution, and to increase the power of computer. Linux, UNIX and Windows 7 are examples of multiprocessing OS.

✡ **Time sharing Operating System:** It allows execution of more than one tasks or processes concurrently. For this, the processor time is divided amongst different tasks. This division of time is

also called **time sharing**. The processor switches rapidly between various processes. After the stipulated time is over, the CPU shifts to next task in waiting, So this type of operating system employs **round robin scheduling technique.** The system switches rapidly from one user to another but still each user feels that it is getting a dedicated CPU time. Virtual Memory techniques are used in this type of operating system. For example, the user can listen to music on the computer while writing an article using a word processing software. The user can switch between the applications and also transfer data between them. Time sharing operating system can be both single user and multiuser. Windows 95 and all later versions of Windows are examples of multitasking OS.

- **Real Time Operating System:** It is a multitasking operating system designed for real time applications like robotics. In this type of operating system, the tasks have to be done within a fixed deadline. System performance is good if task is finished within this deadline. If it is not done, the situation is called Deadline Overrun. Lesser the deadline over run, better is the system efficiency. Hence Real Time operating systems depend not only on the logical result of the computation but also on the time in which the results are produced.
- **Distributed Operating System:** On a network data is stored and processed on multiple locations. The Distributed Operating System is used on networks as it allows shared data/files to be accessed from any machine on the network in a transparent manner. We can insert and remove the data and can even access all the input and output devices. The users feel as if all data is available on their workstation itself.
- **Interactive Operating System:** This is the operating system that provides a Graphic User Interface (GUI) through which the user can easily navigate and interact. The computer responds almost immediately after an instruction has been entered, and the user can enter new instructions after seeing the results of the previous instructions.

Commonly Used Operating Systems

Some of the commonly used operating systems are discussed below:

1. Windows: Microsoft launched Windows 1.0 operating system in 1985 and since then Windows has ruled the world's software market. It is a GUI (Graphic User Interface) and various versions of Windows have been launched like Windows 95, Windows 98, Win NT, Windows XP, Windows 7 and the latest being Windows 8.

2. Linux: Linux is a free and open software which means it is freely available for use and since its source code is also available so anybody can use it, modify it and redistribute it. It can be downloaded from www.linux.org. It is a very popular operating system used and supported by many companies. The defining component of this operating system is the Linux kernel.

3. BOSS (Bharat Operating System Solutions): This is an Indian distribution of GNU/Linux. It consists of Linux operating system kernel, office application suite, Bharateeya OO, Internet browser (Firefox), multimedia applications and file sharing.

4. UNIX: It is a multitasking, multiuser operating system originally developed in 1969 at Bell Labs. It was one of the first operating systems developed in a high level language, namely C. Due to its portability, flexibility and power, UNIX is widely being used in a networked environment. Today, "UNIX" and "Single UNIX Specification" interface are owned and trademarked by The Open Group. There are many different varieties of UNIX, although they share common similarities, the most popular being GNU/Linux and Mac OS X.

5. Solaris: It is a free Unix based operating system introduced by Sun Microsystems in 1992. It is now also known as Oracle Solaris. Solaris is registered as compliant with Single UNIX Specification. It is quite scalable and is used on virtual machines.

Mobile Operating Systems (Mobile OS)

It is the operating system that operates on digital mobile devices like smart phones and tablets. It extends the features of a normal operating system for personal computers so as to include touch screen, Bluetooth, WiFi, GPS mobile navigation, camera, music player and many more. The most commonly used mobile operating systems are – Android and Symbian

Android: It is a Linux derived Mobile OS released on 5th November 2007 and by 2011 it had more than 50% of the global Smartphone market share. It is Google's open and free software that includes an operating system, middleware and some key applications for use on mobile devices. Android applications are quiet user friendly and even one can easily customize the Smartphone with Android OS. Various versions of Android OS have been released like 1.0, 1.5, 1.6, 2. x, 3.0 etc. Most Android phones use the 2.x release while Android 3.0 is available only for tablets. The latest Android version released is 4.2.2. The Android releases have dessert inspired codenames like Cupcake, Honeycomb, Ice Cream sandwich and Jelly Bean.

Symbian: This Mobile OS by Nokia (currently being maintained by Accenture) designed for smartphones. It offers high level of functional integration between communication and personal information management. It has an integrated mail box and it completely facilitates the usage of all Google applications in your smartphone easily. Symbian applications are easy to shut down as compared to Android applications. Various versions like S60 series, S80 series, S90 series, Symbian Anna etc have been released. The latest Symbian releases (Symbian Belle) can support 48 languages.

Language Processors

We know that computer understands instructions in machine code, i.e. in the form of 0s and 1s. It is difficult for us to write computer program directly in machine code. The programs are written mostly in high-level languages, i.e. BASIC, C++, Python etc. A program written in any high-level programming language (or written in assembly language) is called the Source Program or Source Code. The source code cannot be executed directly by the computer. The source code must be converted into machine language to be executed. The program translated into machine code is known as Object Program or Object code. The special translator system software that is used to translate the program written in high-level language (or Assembly language) into machine code is called language processor or translator program. The language processors can be any of the following three types- Assembler, Compiler and Interpreter.

Assembler

The Assembler is used to translate the program written in Assembly language into machine code. The input of Assembler is a source program that contains assembly language instructions. The output generated by assembler is the object code or machine code understandable by the computer.

Compiler

The language processor that translates the complete source program as a whole in onego into machine code is called compiler. Some of the examples are C and C++ compilers. The program translated into machine code is called the object program. The source code is translated to object code successfully if it is free of errors. If there are any errors in the source code, the compiler specifies the errors at the end of compilation with line numbers. The errors must be removed before the compiler can successfully recompile the source code again.

Interpreter

The language processor that translates a single statement of source program into machine code and executes it immediately before moving on to the next line is called an Interpreter. If there is an error in the statement the interpreter terminates its translating process at that statement and displays an error message. Only after removal of the error, the interpreter moves on to the next line for execution.

Utilities

A utility software is one which provides certain tasks that help in proper maintenance of the computer. The job of utility programs is to keep the computer system running smoothly. Nowadays many utility softwares are part of the operating system itself. Even if there is no utility software on your computer, the computer works but with the right kind of utility software loaded, the computer becomes more reliable and even its processing speed increases. Some of the commonly use utility softwares are antivirus, Disk defragmenter, backup, compression etc.

✡ Antivirus

An antivirus is utility software which detects and removes computer viruses. If the software is not able to remove the virus, it is neutralized. The antivirus keeps a watch on the functioning of the computer system. If a virus is found it may alert the user, flag the infected program or kill the virus. Some of the common types of viruses are:

- **Boot Sector Virus:** A boot sector virus displaces the boot record and copies itself to the boot sector i.e. where the program to boot the machine is stored. So first the virus is loaded on to the main memory and then the operating system. Whenever a new disk is inserted the virus copies itself to the new disk. The antivirus overwrites the correct boot record on the infected boot sector and also cleans the bad sectors.
- **File Virus:** A file virus generally attacks executable files. They can attach to various locations of the original file, replace code, fill in open spaces in the code, or create companion files to work with an executable file. Most of the file viruses are memory resident and wait in the memory until the user runs another program. While another program is running, the virus replicates.
- **Macro Virus:** This virus infects an important file called normal.dot of MS Word. As soon as the application is opened the virus gets activated. It damages the formatting of documents and even may not allow editing or saving of documents.
- **Trojan Horse:** It is a code generally hidden in games or spreadsheets. Since they are hidden, the program seems to function as the user wants but actually it is destroying the program. A Trojan horse does not require a host program to embed itself. It is a complete program. Its main objective is to cause harm to the data. They can create bad sectors on the disk, destroy file allocation tables and cause the system to hang.
- **Worm:** Worm is a program capable of replicating itself on a computer network. A worm also does not require a host as it is a self contained program. They generally travel from one computer to another across communication links on a network. They generally disrupt routine services.

✡ Disk Defragmenter

The memory is used in small chunks randomly. Sometimes when a memory chunk of appropriate size is not available, the operating system breaks or fragments the files resulting in slower access to files. A disk defragmenter scans the hard disk for fragmented files and brings all the fragments together.

✡ Backup Utility

This utility is used to create the copy of the complete or partial data stored in a disk or CD on any other disk. In case the hard disk crashes or some other system failure occurs, the files can be restored using backup software.

✡ Compression Utility

This utility is used to compress large files. Compression is useful because it helps reduce resources usage and the file transmission on the network becomes easier.

✡ Disk Cleaner

This utility scans for file that have not been accessed/used since long. Such files might be occupying huge amount of memory space. In that case the Disk Cleaner utility prompts the user to delete such

files so as to create more space on the disk. If the files are important, the user might take a backup before deleting them.

✡ File Management Tools

This utility helps the user in storing, indexing, searching and sorting files and folders on the system. The most commonly used tool is the Windows Explorer and Google Desktop.

Application Software

An application software is bought by the user to perform specific applications or tasks, say for example making a document or making a presentation or handling inventory or managing the employee database. An application software can be of two types – General Purpose Application Software and Customized Application software.

General Purpose Application Software

Some of the application software is made for the common users for day to day applications and uses. These are also referred as Office Tools. The users may use them in the manner they want. Some of the popular types of general purpose application software are discussed below:

✡ **Word Processor:** Word processor is a general purpose application software used to create documents. It allows us to create , edit and format documents. We can use different types of fonts of various sizes; underline or make bold a certain part of the text. We can add clipart and other graphics into the document. Popular examples of Word processing software are Writer (Open Office) and Microsoft Word.

We use word processing software for various uses like writing a simple document to designing special art effects. Since we can attach images and different shapes, can use different colors, even a poster can be designed using word processing software. Features like Mail Merge, Macro has further enhanced the word processing software and made it very useful.

✡ **Presentation Tools:** Presentation tools is a general purpose application software that lets us create presentations on any topic. We can not only create a presentation and add slides into that but also can use different types of background, fonts, animations, audio, video, etc. We can add clipart and other graphics into our document. Even audio video files can be added on to the presentations. Popular examples of Presentation tools software are Impress (open office) and Microsoft Power Point.

✡ **Spreadsheet Packages:** Spreadsheet is a general purpose application software that lets us create and store data in tabular form. Both text and numerical values can be entered in that tables known as a spreadsheet. We can not only create a document and add data into that but also can create different types of charts and graphs based upon the numerical data stored in that page. All common mathematical and statistical formulae can be used on the numeric data. Popular examples of Spreadsheet software are Calc (Open Office) and Microsoft Excel.

✡ **Database Management System:** Database Management System is general purpose application software that lets us create computer programs that control the creation, maintenance, and the use of database for an organization and its end users. We can not only store data but can also manage data in a database. We can also import and export the data to many formats including Excel, Outlook, ASCII, dBase, FoxPro, Oracle, SQL Server, ODBC, etc. Popular examples of Database Management System are Base (Open Office) and Microsoft Access.

Customized Software

Customized Software is one which is tailor made as per the user's requirement. Such type of software is customer specific. It is made keeping in mind the individual needs of the user and so are also referred

as Domain Specific Tools. Such software cannot be installed and used by any other user/customer since the requirements may differ. Some examples of customized software are discussed below:

- **Inventory Management System & Purchasing System:** Inventory Management System is generally used in departmental stores or other organizations to keep the record of the stock of all the physical resources. For Example, in a Computer store, it keeps record of the number of computers, printers, printing sheet, printer cartridge available. It also helps to place purchase orders, bills, invoices etc. Various reports as to position of stock, sales made in a particular period, profit earned etc. can be generated.
- **School Management System:** School Management System (sometimes called a School Information System or SIS) is a system that manages all of a school's data in a single, integrated application. Having all of the information in a single system allows schools to more easily connect data together. For example, when viewing a student's record, the user can follow a link to the student's class, and from there a link to the student's teacher, and from there a link to the teacher's other classes, and so on
- **Payroll System:** Payroll Management System software is used by all modern organizations to keep track of employees of the organization who receives wages or salary. All different payment amounts are calculated by the payroll software and the record is maintained. The software keeps track of personal records of employees viz. name, address, date of birth, qualification, date of joining etc. It also keeps track of professional record viz. allowances, perks, income tax, insurance etc. Different reports, pay slips etc can be generated through this software.
- **Financial Accounting:** Financial accounting System is used to prepare accounting information, maintain different accounts ledger, and account books. It also helps an organization to make budget.
- **Hotel Management:** Hotel management software refers to management techniques used in the hotel sector. These can include hotel administration, accounts, billing, marketing, housekeeping, front office or front desk, food and beverage management, catering and maintenance. Even advance bookings can be made through this software. Customers can have a look at the hotel and the rooms before making bookings. At any point of time the room availability, tariff for each type of room and even booking status can be checked.
- **Reservation System:** Reservation System is software used to book (reserve) air flights, railway seats, movie tickets, tables in a restaurant, etc. In the case of a booking system, the inputs are booking requests. The processing involves checking if bookings are possible, and if so making the bookings. The outputs are booking confirmations/rejections.
- **Weather Forecasting system:** This software makes it possible to forecast the weather for days and even months in advance. The detailed weather reports can also be generated.

Open Source Concepts

Software are mainly categorised into the following categories based on their licenses:

1. Proprietary
2. Shareware
3. Freeware
4. Open source
5. Free Software

✡ Proprietary

We pay a supplier for a copy of the software which these days may be supplied on physical media (disks) or downloaded from the Internet. We get the permission to use the software on one or sometimes more than one machines. Examples of this type of software include Microsoft Office and Microsoft Windows.

✡ Shareware

Shareware is basically a software for trial purpose that the user is allowed to try for free, for a specified period of time. It is usually downloaded from the Internet. When the trial period ends, the software must be purchased or uninstalled.

✡ Freeware

Freeware software is free of cost and is usually bundled up with some operating system or any other software. Examples of freeware include Microsoft Internet Explorer which comes bundled up with any Microsoft operating system. The author of the freeware software is the owner of the software, though people may use it for free. The source code is not available, so no modifications can be done. Freeware should not be mistaken with Open Source Software or Free Software.

✡ Open source

Open Source Software (OSS) is the software which gives the users freedom to run/use the software for any purpose and in any manner. They can be used, modified and even redistributed. In simple terms it can be freely used but it may not be free of charge. The source code is freely available to the customer. Python, Tux Paint etc are examples of Open Source Software.

✡ Free Software

This type of software is freely accessible and can be freely used, modified, copied or distributed by anyone. And no licence fee or any other form of payment need to be made for a free software. The source code is also accessible in case of free softwares.

Data Representation in Computers

After studying this chapter the student will be able to:

**Learn about binary, octal, decimal and hexadecimal number systems*
**Learn conversions between two different number systems*
**Understand internal storage encoding of characters: ASCII, ISCII and UNICODE*

Binary Representation of Data

In order to work with data, the data must be represented inside the computer. Digital computers represent data by means of an easily identified symbol called a digit.

Numbering Systems

Each number system has a base also called a Radix. A decimal number system is a system of base 10; binary is a system of base 2; octal is a system of base 8; and hexadecimal is a system of base 16. What are these varying bases? The answer lies in what happens when we count up to the maximum number that the numbering system allows. In base 10, we can count from 0 to 9, that is,10 digits.

Number System	Base	Symbols used
Binary	*2*	*0,1*
Octal	*8*	*0,1,2,3,4,5,6,7*
Decimal	*10*	0,1,2,3,4,5,6,7,8,9
Hexadecimal	*16*	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F where A = 10; B = 11; C = 12; D = 13; E = 14; F = 15

Converting a number from one Base to another:

Other (Binary/Octal/Hexadecimal**) to Decimal**

Method to convert other to Decimal:

1. Start at the rightmost bit.
2. Take that bit and multiply by B^n, where B is the base of other number system and n is the current position beginning at 0 and increasing by 1 each time.
3. Sum each terms of product until all bits have been used.

Example

Convert the Binary number 101011 to its Decimal equivalent.

$1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0$

$32 + 0 + 8 + 0 + 2 + 1 = (43)_{10}$

Convert the Octal number 321 to its Decimal equivalent.

$3 * 8^2 + 2 * 8^1 + 1 * 8^0$

$192 + 16 + 1 = (209)_{10}$

Decimal to Other (Binary/Octal/Hexadecimal)

Method to convert a Decimal number into its Binary equivalent:

1. Divide the decimal number by B.
2. Take the remainder and record it on the side.
3. Divide the quotient by B
4. REPEAT UNTIL the decimal number cannot be divided further.
5. Record the remainders in reverse order and you get the resultant binary/octal/hexadecimal number.

Example

Convert the Decimal number 125 into its Octal equivalent.

125 / 8 = 15 5
15/ 8 = 1 7
1/8 =0 1

Answer: $(175)_8$

Table to remember

Decimal	Binary	Hexadecimal	Octal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17
16	10000	10	20

Convert from binary to octal: For this conversion make the group of three digits from right to left before decimal & left to right after decimal then assign the specific octal value. (Given in the table above)

$110101000.101010_{(2)}$->$X_{(8)}$

| 3 | | 3 | | 3 | | 3 | | 3 |

110 101 000 .101 010

|| || || || ||

\/ \/ \/ \/ \/

6 5 0 . 5 2 (See that in the array $110_{(2)}$ corresponds to $6_{(8)}$)

$110101000.101010_{(2)}$->$650.52_{(8)}$

Convert from binary to hexadecimal: This conversion make the group of four digits from right to left before decimal & left to right after decimal then assign the specific Hexadecimal value. (Given in the table above)

$110101000.101010_{(2)}$->$X_{(16)}$

| 4 | | 4 | | 4 | | 4 | | 4 |

0001 1010 1000 .1010 1000

|| || || || ||

\/ \/ \/ \/ \/

1 A 8 . A 8

$110101000_{(2)}$->$1A8.A8_{(16)}$

Convert from hexadecimal to octal and binary: In this conversion write the binary of specific digit. For Octal three digit binary & for Hexadecimal four digit binary.

Convert from octal to binary

$650.52_{(8)}$->$X_{(2)}$

6 5 0 . 5 2

|| || || || ||

\/ \/ \/ \/ \/

110 101 000 . 101 010

$650.52_{(8)}$->$110101000.101010_{(2)}$

Convert from hexadecimal to binary

$1A8.A8_{(16)}$->$X_{(2)}$

1 A 8 . A 8

|| || || || ||

\/ \/ \/ \/ \/

0001 1010 1000 .1010 1000

Binary Representation of Integers

Binary number can be represented only by using 0's and 1's, but can not use the sign (-) to denote the negative number or sign (+) to denote the positive number. So it must be either 0 or 1. There are three methods to represent binary number. They are

(i) Sign and magnitude method
(ii) One's complement method
(iii) Two's complement method

Sign and magnitude method

In this method, first bit is considered as a sign bit. Here positive number starts with 0 and negative number starts with 1.

Example

25

25/2 = 12	1
12/2 = 6	0
6/2 = 3	0
3/2 = 1	1
1/2 = 0	1

So the binary number is $(11001)_2$. If we take the size of the word is 1 byte, then the number 25 will be represented as 00011001

Suppose, if the number is -25, and then it will be represented as 10011001

One's Complement Method

In this method, the positive number is represented as same as the binary number. If the number is negative, then we need to find one's complement of a binary number. The one's complement of a binary number will replace every 0 with 1 and vice- versa.

Example

(i) Represent 86 in one's complement method (I byte representation)

86/2 =43 0
43/2 =21 1
21/2 =10 1
10/2 =5 0
5/2 =2 1
2/2 =1 0
1/2 =0 1

The binary number is 1010110
1 byte representation of number 86 is 01010110

(ii) Represent -86 in one's complement method (I byte representation)
The given number is negative; hence we need to calculate one's complement
One's complement of 01010110 is 10101001 (convert 1 into 0 and 0 into 1)
Thus, the 1 byte representation of number -86 is 10101001.

Two's Complement method

In this method, the positive number is represented as the binary number. If the number is negative, then we need to calculate two's complement of a binary number. The two's complement of a binary number is calculated by adding 1 to its one's complement.

Example

(i) Represent -54 two's complement method (I byte representation)

54/2 =27 0
27/2 =13 1
13/2 =6 1
6/2 =3 0
3/2 =1 1
1/2 =0 1

The binary number is 110110
Hence, the 1 byte representation is 00110110
The given number is negative; hence we need to calculate two's complement.
One's complement of 00110110 is 11001001 (convert 1 into 0 and 0 into 1)
Add 1 to one's complement

11001001 (1+1=2, binary equivalent =10)
+ 1
11001010
Thus, 1 byte representation of number -54 is 11001010

Representing Characters

A computer can handle numeric and non numeric data like letters, punctuation marks and other special characters. Some predefined codes are used to represent numeric and non numeric characters. Some of the standards are discussed below:

ASCII

ASCII stands for American Standard Code for Information Interchange. ASCII-7 can represent 128 characters. Out of 7 bits, 3 are zone bits and 4 are numeric bits. ASCII-8 can represent 256 characters. It is an extended form of ASCII-7.

ISCII: (Indian Standard Code for Information Interchange)

A lot of efforts have gone into facilitating the use of Indian languages on computers. In 1991, the Bureau of Indian Standards adopted the ISCII. It is an 8 bit code which allows English and Indian Scripts alphabets to be used simultaneously. Characters coded in ISCII need 8 bits for each character.

Unicode

Unicode is a new universal coding standard adopted by all new platforms. It is promoted by Unicode Consortium which is a non profit organization. Unicode provides a unique number for every character irrespective of the platform, program and the language. It is a character coding system designed to support the worldwide interchange, processing, and display of the written texts of the diverse languages.

Microprocessors and Memory Concepts

After studying this session students will be able to:
**Learn about a microprocessor and its characteristics*
**Classify the microprocessors – RISC, CISC and EPIC*
**Learn the various units of memory*
**Understand the memory types- Primary memory and Secondary memory*
**Learn about various input- output ports /connections*

Microprocessor

We studied that the Central processing unit processes data inside the computer. It interprets all the instructions given to it and carries out these instructions. A microprocessor is a Central Processing Unit (CPU) on a single chip. It is a multipurpose programmable device constructed using Metal Oxide Semiconductor (MOS) technology. In 1971 Intel Corporation fabricated the first microprocessor- 4004. It could do only add and subtract operations and that too it could process only 4 bits at a time. But Intel 4004 powered one of the first portable electronic calculators and since then the microprocessor technology has come a long way. Today we have processors with which can process upto 128 bits at a time at the speed of billion instructions per second. Let us now have a look at the characteristics of microprocessors:

- **Instruction Set:** It is the set of instructions that the microprocessor executes
- **Word Length:** The number of bits processed in a single instruction is called word length or word size. The word size is directly proportional to the processing power of the CPU. During the processing, the internal general purpose registers hold data. So if internal registers can hold data upto 8 bits, the word length is 8 bits. If it can process 16 bits at a time, then the internal registers can hold upto 16 bits at a time and so on. Hence a 32 bit processor is about 4 times faster than an 8 bit processor. Examples of word lengths are 16 bit, 32 bit, 64 bit. The terms 16-bit CPU, 32-bit CPU, 64-bit CPU are used very often while talking about CPUs. Now we know that these terms mean the maximum number of bits a given CPU can handle at a time.
- **System Clock Speed:** The microprocessor's pace is controlled by the System Clock. The System Clock is an electronic circuit that generates pulses which are measured in million of cycles per second (MHz). The number of pulses generated by the clock per unit of time is its Clock speed. Each microprocessor is characterized by its clock speed. Nowadays microprocessors have clock speed of several GHz. The CPU uses this clock speed to control sequencing and execution of various operations in the computer.

Classification of Microprocessors

Apart from the width of data (word length) that the microprocessors can process at a time, the classification is also based on the architecture i.e. Instruction Set of the microprocessor. While studying about CPUs, we come across two abbreviations CISC and RISC.

- **RISC:** It stands for Reduced Instruction Set Computer. It is a type of microprocessor architecture that uses a small set of instructions of uniform length. These are simple but primitive instructions which execute in one clock cycle. For this reason, RISC chips are less complex and also less expensive

to produce. The instructions are of uniform length which interface with about 32-36 registers. The program size in case of RISC architecture is more but more memory cycles are needed to access data. To reduce the number of memory cycles, RISC keeps the necessary data in the processor itself. The drawback of RISC design is that the computer must combine or repeat operations to complete a large program consisting of many processing operations. Since instructions are simple , RISC processors are relatively simple to design. Examples of RISC processor is SPARC, POWER PC etc.

- **CISC:** It stands for Complex Instruction Set Computer. A CISC chip such as Intel Pentium provides programmers with hundreds of instructions of variable sizes, and the processing circuitry includes many special purpose circuits that carry out these instructions at high speeds. These instructions interface with memory in multiple mechanisms with complex addressing modes. In this case the program size is reduced and hence lesser number of memory cycles are required to execute the instruction. So fewer general purpose registers(8-12) are present in CISC processors. Also less number of memory cycles result in faster execution of the program.
- **EPIC:** It stands for Explicitly Parallel Instruction Computing. It is a computer architecture that combines the best feature of both RISC and CISC. It does not use instructions of any fixed length but rather aims at parallel processing of instructions. It uses a bundle of complex instructions that in addition to basic instruction also contain information on how to run the instruction in parallel with other instructions. This greatly increases the efficiency of an EPIC processor. IA-64

(Intel Architecture-64) is Intel's first 64 bit processor based on EPIC.

Memory Concepts

Memory is one of the most important components of a computer system as it stores data and instructions. Every memory chip contains thousands of memory locations. In the computer, the data is stored in the form of bits and bytes. A **bit** is the smallest storage unit of memory. A **nibble** is a collection of 4 bits. Eight bits combined together to form a single **byte**, which in turn represents a single character. Other units of memory are KB (Kilobyte), MB (Megabyte), GB (Gigabyte) TB(Terabytes), PB (Petabyte), EB (Exabytes), ZB (Zettabytes) and YB (Yottabytes). Every higher unit is equal to 2^{10} of the previous unit.

The smallest unit is bit, which mean either 0 or 1.

1 bit	= 0 or 1
1 Byte	= 8 bit
1 Nibble	= 4 bit
1 Kilo Byte	= 1024 Byte= 2^{10} Byte
1 Mega Byte	= 1024 KB= 2^{10} KB
1 Gega Byte	= 1024 MB= 2^{10} MB
1 Tera Byte	= 1024 GB= 2^{10} GB
1 Peta Byte	=1024 TB= 2^{10} TB
1 Exa Byte	=1024 PB= 2^{10} PB
1 Zetta Byte	= 1024 EB= 2^{10} EB
1 Yotta Byte	= 1024 ZB= 2^{10} ZB

The computer memories can be divided into following categories:

- Primary Memory
- Secondary memory
- Cache Memory

Primary Memory

Primary memory or main memory is a Metal Oxide Semiconductor (MOS) memory used for storing program and data during the execution of the program. It is directly accessible to CPU. Memories can be both read from and written to are called **read/write memories**. On the other hand memories that have data or program permanently stored onto them and hence can be only read from are called **Read Only memories**. Broadly primary memory can be of two types – RAM (Random Access Memory) and ROM (Read only memory).

Random Access Memory (RAM)

In case of RAM, the memory can be accessed from any desired location randomly. That means without searching the entire memory, any location can be accessed in the same amount of time. The instructions and data that we input into the computer are stored in the RAM of the Computer. It is a read/write memory, so data can be both read from and written to the RAM. It is a volatile memory and loses its contents when the power is switched off or interrupted. Nowadays RAMs are available in gigabytes. The normal memory access time of a RAM is 20-80 ns. RAM can be broadly classified into two categories: Dynamic RAM (DRAM) and Static RAM (SRAM).

Dynamic RAM (DRAM): It consists of a transistor and a capacitor that stores electric charge. The DRAMs are physically smaller, cheaper and slower memories. They are slower because the data stored in them needs to continuously refreshed and this consumes lot of processor time. Each refresh operation takes several CPU cycles to complete. This is because a capacitor tends to loose charge over a period of time which needs to be refreshed again and again. DRAM is used in primary storage areas and is available in various forms as EDORAM (**E**xtended **D**ata **O**utput RAM), SDRAM (**S**ynchronous DRAM) and DDR SDRAM.

Static RAM: This type of RAM is large in physical size but is an expensive and faster memory. It is faster because it is made up of flip flops to store data and these flip flops do not require any refreshing. Static RAM is also volatile and is easier to use as compared to dynamic RAM. These are used in specialized applications.

Read only memory (ROM)

As the name suggests, a ROM is a type of memory that can perform read operation only. The contents of ROM are written by the manufacturer and come along with the computer. We cannot change its contents or write something else on it. Data is written on to the ROM at the time of its manufacture and it cannot be changed thereafter. It is a non-volatile memory, which means that contents stored in it are not lost even when the power to the computer is switched off. ROMs are used in applications where the information once written, need not be altered. They hold certain essential instructions such as interrupt service routines or a monitor program controlling the machine. Instructions that are needed to start the computer are also stored in the ROM. ROMs are slower as compared to RAMs and are available in various types –

- **Programmable Read Only Memory (PROM):** This type of ROM can be programmed even after its manufacture using a PROM programmer circuit. But once a PROM is programmed, it becomes just like ROM i.e. it cannot be changed.
- **Erasable Programmable Read Only Memory (EPROM):** In this type of ROM, the contents can be erased and the memory can be reprogrammed. To erase the data, an EPROM is exposed to

ultraviolet light and then it can be reprogrammed using a PROM programmer circuit. When the EPROM is in use, then it behaves like a ROM, that means the information can only be read.

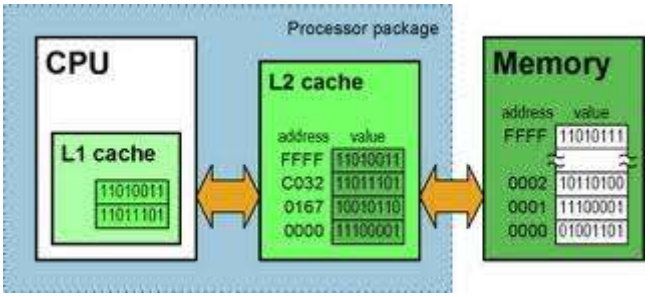
✡ **Electrically Erasable Programmable Read Only Memory (EEPROM):** The contents of this type of ROM can be erased and then reprogrammed using electric signals. This makes it an excellent back up for RAM whose contents are lost when the power is switched off. When the power is returned, the contents of EEPROM are copied back into the RAM and the computer continues working without any data loss. Nowadays RAMs and EEPROMs are integrated in a single chip.

Cache Memory

Cache memory is a special high speed memory made up of high speed static RAMs. It is used to hold frequently accessed data and instructions. We know that the processing speed of CPU is much more than the main memory access time of the computer. This means the CPU has to wait for a substantial amount of time. Alternatively we have the cache memory which is a small, expensive but fast memory that is placed between the CPU and the main memory. Whenever some data is required, the CPU first looks into cache. If data is found, we call it a **cache hit** and the information is transferred to the CPU. In case of a miss, the main memory is accessed. Memory caching proves to be efficient because most programs repeatedly access the same data and instructions, so access of frequently used data becomes very fast with cache memory. There are two types of cache memory:

✡ **L1 cache:** It is small and is built inside the CPU. It is fast as compared to L2 cache

✡ **L2 cache:** It is large but slower and is mounted on the motherboard



Secondary Memory

The major limitation of primary memory is that it has limited storage capacity and is volatile. To overcome this limitation we have secondary memory storage devices. This type of memory is also called external memory. It refers to the various storage media on which a computer can store data and programs. It is an additional storage, not part of the main computer.

The Secondary storage media can be fixed or removable. *Fixed Storage media* is an internal storage medium like hard disk that is fixed inside the computer. *Removable storage media* is a data storage medium that is portable and can be taken outside the computer.

Why do we need Secondary Memory?

Secondary memory is needed because of the following reasons:

a. **For permanence:** As the RAM is volatile, i.e. it loses all information when the electricity is turned off, something is needed for permanence. Secondary storage devices serve this purpose. They do not lose data when electricity is turned off.

b. **For portability:** Secondary storage, like the CDs, flash drives can be used to transport data from one computer to another.

Secondary Storage Media

There are the following main types of storage media.

a. Magnetic b. Optical c. Solid State

Magnetic storage media: Examples of magnetic storage media are hard disks, floppy disks and magnetic tapes. Magnetic media is coated with a magnetic sensitive layer and this layer is magnetized in clockwise or anticlockwise directions, which then are interpreted as binary 1s and 0s at reading.

Floppy Disk (Diskette): A floppy disk is a flexible disk made up of mylar with a magnetic coating on it. It is packaged inside a protective plastic envelope. These were one of the oldest type of portable storage devices that could store up to 1.44 MB of data but now they are no longer in use.

Hard disk: A hard disk consists of one or more circular disks called platters which are mounted on a common spindle. Each surface of a platter is coated with a magnetic material. Both surfaces of each disk are capable of storing data except the top and bottom disk where only the inner surface is used.

The information is recorded on the surface of the rotating disk by magnetic read/write heads. These heads are joined to a common arm known as access arm. This arm moves over the surface of the rotating. Information is recorded on each of these disks in the form of concentric circles called tracks which are further divided into sectors. Hard drives however, are not very portable and are primarily used internally in a computer system. But external hard disks are also available as a substitute for portable storage. Today the hard disks have the storage capacity of several gigabytes to terabytes.

Optical storage media

On an optical storage media information is stored and read using a laser beam. The data is stored as a spiral pattern of pits and ridges denoting binary 0 and binary 1. Examples of optical media are CDs, DVDs etc.

Compact Disk: A compact disk or CD can store approximately 650 to 700 megabytes (MB) of data. We must have a CD drive in our computer to read them. The bits (0 and 1) are encoded as transitions between raised ridges and etched pits, which are lined up in a spiral like pattern. This pattern is then stamped into a 1.2-mm clear polycarbonate disc (a CD), which is then covered with a super thin coating of reflective metal (usually aluminum or gold) and a label. To read the data, an infrared laser is beamed through the CD's polycarbonate substrate. The wavelength of light that bounces off the mirror-like reflective backing is then measured. A pit scatters the light and the ridge reflects the light. Since pits and ridges pass different amounts of light, the fluctuations in the reflected beam are then translated back into the original ones and zeros.

There are three types of CDs:

CD- ROM: It stands for Compact Disk - Read Only Memory and data is written on these disks at the time of manufacture. Thereafter this data cannot be changed but can only be read by a laser beam in the form of a continuous spiral. CD- ROMs are used for text, audio and video distribution like games, encyclopedias and application softwares.

CD-R: It stands for Compact Disk- Recordable. Data can be recorded on these disks but only once. So we can write data on these disks through a read/write CD drive but after that the disk cannot be erased/modified.

CD-RW: It stands for Compact Disk-Rewritable. It can be read or written multiple times. But a CD-RW drive needs to be installed on your computer.

DVD: It stands for Digital Versatile Disk or Digital Video Disk. It looks just like a CD and use a similar technology as that of the CDs discussed above but employ a shorter wave length red laser that permits a narrower beam. This allows tracks to be spaced closely enough to store data that is more than six times the CD's 700MB capacity. It is a significant advancement in portable storage technology. DVDs consist of two half thickness (0.6-mm) CD-like discs glued back-to-back. This protects the delicate reflective coating as it is on the inside of the disc. Also it makes possible to have double sided
DVDs—where data can be stored on each half disc.

A DVD holds 4.7 GB to 17 GB of data. That means a complete movie can be stored on one side of a DVD. Like CDs DVDs also come in three varieties –

- DVD- ROM
- DVD- R
- DVD-RW

Blue Ray Disk: This is the latest optical storage media to store high definition audio and video. It looks like a CD or DVD but can store up to 27 GB of data on a single layer disk and up to 54 GB of data on a dual layer disk. Where CDs or DVDs use red laser beam, the blue ray disk uses a blue laser to read/write data on a disk. As the wavelength of the blue ray is shorter, more data per unit area can be stored on the disk. This is because due to shorter wavelength, it is possible to focus the laser spot with greater precision. Hence data can be packed more tightly. Blue-ray Disc (BD) was developed to enable recording, rewriting and playback of high-definition video (HD), as well as storing large amounts of data.

Solid State Memories

The term 'solid-state' essentially means 'no moving parts'. Hence Solid-state storage devices are based on **electronic circuits** with **no moving parts** (no reels of tape, no spinning discs, no laser beams, etc.) Solid-state storage devices store data using a special type of memory called flash memory. SSD, Solid-state drive (or flash memory) is used mainly in digital cameras, pen drives or USB flash drives.

Pen Drives: Pen Drives or Thumb drives or Flash drives are the recently emerged portable storage media. It is an EEPROM based flash memory which can be repeatedly erased and written using electric signals. This memory is coupled with a USB connector through which it can be plugged into the computer. They have a capacity smaller than a hard disk but greater than a CD.

Input Output Ports and Connections

Let us look at the back of a CPU. Computer ports are the points where external devices or peripherals connect to a computer. These ports are available at the rear or front of the computer. We connect peripherals to the computer with a cable that attaches to one of the ports. A port's main function is to

act as a point of attachment where the cable from the peripheral device plugs into the system unit, allowing data to flow from the peripheral device. Some of the common sockets/ports are power socket for connecting power cable, PS2 ports for connecting Mouse & Keyboard, USB Port for connecting USB devices such as mouse, keyboard, printer, pen drive etc. and VGA port for connecting Monitor/Screen.

Some ports are discussed in detail below:

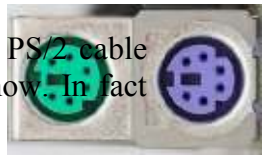
Serial Port

Through a serial port data is transmitted travels one bit at a time through a single wire. The data transmission speed is quite slow. Serial ports are commonly known as communication (COM) ports or RS232C ports and connect devices like mouse and modem. These ports are rarely used these days.

Parallel port

These ports were earlier used to connect printers to the computer system. A parallel port can send 8 bits (1 byte) at a time simultaneously (in parallel). Hence data transmission is faster through these ports. Parallel ports are used to connect printers, scanners, CD writers etc.

PS/2 Port



This is a round port for plugging in keyboard or mouse. It has a PS/2 cable with a mini DIN connector. These ports are becoming obsolete now. In fact some systems these days do not have PS/2 ports.

USB Port

A USB (Universal Serial Bus) port is a standard cable connection interface available on personal computers and some other electronic devices for data communication. It is a single, low cost, plug n play connector. The operating system automatically detects the device connected through the USB port. USB ports have become very popular these days as they connect many different devices to the computer these days. Most computers have USB ports on front, back and/or sides of system unit. Flash drives, digital cameras, printers, scanners are some of the devices that often connect through the USB port.

Infrared Port

In this type of port, data is transmitted through Infrared waves. For infrared transmission the device and the computer both must have infrared ports. These allow computers and peripherals to communicate serially over an Infrared link rather than over cables. The remote control of our TV sets uses the same technology. For wireless data communication between computers and various peripheral devices we use infrared ports.

Bluetooth Port

Bluetooth is used to connect mobile phones, computers and PDAs using a short range wireless connection. This technology uses radio waves to transmit data between any two devices. The devices that are Bluetooth enabled contain a small transceiver chip that allows them to communicate with other

Bluetooth enabled computer or device. Data can be exchanged at the rate of about 2 megabit per second.

Fire wire Port

FireWire® ports are forms of a serial port that make use of FireWire® technology to transfer data rapidly from one electronic device to another. The FireWire® port has the ability to interact with a number of different devices since it provides a single plug and socket connection for all devices. A FireWire® port can provide an ideal way to connect a scanner and digital camera/camcorder to a computer system as the data transfer is relatively faster than on USB and also results in excellent quality.

UNIT-2
Introduction to C++

C++ CHARACTER SET

Character set is asset of valid characters that a language can recognize . A character can represents any letter, digit, or any other sign . Following are some of the C++ character set.

LETTERS	A to Z and a to z
DIGITS	0 -9
SPECIAL SYMBOLS	+ -* ^ \ [] {} = != < > . ‘ ‘ ; : & #
WHITE SPACE	Blankl space , horizontal tab (- >), carriage return , Newline, Form feed.
OTHER CHARACTERS	256 ASCII characters as data or as literals.

TOKENS:

The smallest lexical unit in a program is known as token. A token can be any keyword,Identifier,Literals, Puncutators, Operators.

KEYWORDS :

These are the reserved words used by the compiler. Following are some of the Keywords.

auto	continue	float	new	signed	volatile
short	long	class	struct	else	inline
delete	friend	private	typedef	void	template
catch	friend	sizeof	union	register	goto

IDENTIFIERS:

An arbitrary name consisting of letters and digits to identify a particular word.C++ is case sensitive as nit treats upper and lower case letters differently. The first character must be a letter . the underscore counts as a letter

Pen time580 s2e2r3 _dos _HJI3_JK

LITERALS:

The data items which never change their value throughout the program run. There are several kind of literals:

- Integer constant
- Character constant
- Floating constant
- String constant.

Integer constant :

Integer constant are whole numbers without any fractional part. An integer constant must have at least one digit and must not contain any decimal point. It may contain either + or _. A number with no sign is assumed as positive.

e.g 15, 1300, -58795.

Character Constant:

A character constant is single character which is enclosed within single quotation marks.

e.g ‘ A’

Floating constant:

Numbers which are having the fractional part are referred as floating numbers or real constants. it may be a positive or negative number. A number with no sign is assumed to be a positive number.

e.g 2.0, 17.5, -0.00256

String Literals:

It is a sequence of letters surrounded by double quotes. E.g "abc".

PUNCTUATORS:

The following characters are used as punctuators which are also know as separators in C++

[ ] { } ( ) , ; : * ……. = #

Puntuator	Name	Function
[]	Brackets	These indicates single and multidimensional array subscripts
()	Parenthesis	These indicate function calls and function parameters.
[ ]	Braces	Indicate the start and end of compound statements.
;	Semicolon	This is a statement terminator.
:	Colon	It indicates a labeled statement
*	Asterisk	It is used as a pointer declaration
…	Ellipsis	These are udes in the formal argument lists of unction prototype to indicate a variable number of arguments.
=	Equal to	It is used as an assigning operator.
#	Pound sign	This is used as preprocessor directives.

OPERATORS:

These are those lexical units that trigger some computation when applied to variables and other objects in an expression. Following are some operators used in C++

Unary operators: Those which require only one operand to trigger. e.g. & , + , ++ , -- ! .

Binary operators: these require two operands to operate upon. Following are some of the Binary **operators.**

Arithmatic operators :

+	Addition
_	substraction
A*	Multiplication
/	Division
%	Remainder.

Logical Operators :

&& - logical AND || - Logical OR

Relational Operator:

< less than
a> Greater than
<= Less than equal to.
>= greater than equal to.
== equal to.
!= not equal to.

Conditional operator: ? (question) : (colon)

Assignment Operator:

= assignment operator
*= Assign Product.
/= Assign quotient
%= assign Remainder
&= Assign bitwise AND
^= Assign bitwise XOR.
|=Assign bitwise OR

Conditional operator (?)

The conditional operator evaluates an expression returning a value if that expression is true and a different one if the expression is evaluated as false. Its format is:

condition ? result1 : result2
e.g 7==5 ? 4 : 3 // returns 3, since 7 is not equal to 5.

Comma operator (,)

The comma operator (,) is used to separate two or more expressions that are included where only one expression is expected. When the set of expressions has to be evaluated for a value, only the rightmost expression is considered.

For example, the following code:

```
a = (b =3 , b +2 );
```

Would first assign the value 3 to b, and then assign b+2 to variable a. So, at the end, variable a would contain the value 5 while variable b would contain value 3.

Explicit type casting operator

Type casting operators allow you to convert a datum of a given type to another. There are several ways to do this in C++. The simplest one, which has been inherited from the C language, is to precede the expression to be converted by the

new type enclosed between parentheses ( ) :

```
int i;
float f =3014;
i = ( int ) f;
```

The previous code converts the float number 3.14 to an integer value (3), the remainder is lost. Here, the typecasting operator was (int). Another way to do the same thing in C++ is using the functional notation: preceding the expression to be converted by the type and enclosing the expression between parentheses:

```
i  = int (f );
```

Both ways of type casting are valid in C++.

sizeof()

This operator accepts one parameter, which can be either a type or a variable itself and returns the size in bytes of that type or object:

```
a= sizeof (char);
```

This will assign the value 1 to a because char is a one-byte long type.
The value returned by sizeof is a constant, so it is always determined before program execution.

Input Output (I/O) In C++

The cout Object:

The cout object sends to the standard output device. cout sends all out put to the screen i.e monitor.

The syntax of **cout** is as follows:

```
cout<< data;.
```

e.g

```
cout<< a ;              ( here a can be any variable)
```

The cin operator :

The cin operator is used to get input from the keyboard. When a program reaches the line with cin, the user at the keyboard can enter values directly into variables.

The syntax of **cin** is as follows:

```
cin>> variablename
```

e.g

```
cin>> ch;      ( here ch can be any variable)
```

- ***Basic structure of a C++ program:***

Following is the structure of a C++ program tht prints a string on the screen:

```
#include<iostream.h>
 void main ()
{
cout<<" Study material for Class XI";
}
```

The program produces following output:

Study material for Class XI

The above program includes the basic elements that every C++ program has. Let us check it line by line

#include<iostream.h> : This line includes the preprocessor directive include which includes the header file iostream in the program.

void main () :this line is the start of compilation for this program. Every C++ programs compilation starts with the **main (). void** is the keyword used when the function has no return values.

{ : this is the start of the compound block of main ().

cout<<" Study material for class XI";: this statement prints the sequence of string **" Study material for class XI"** into this output stream i..e on monitor.

Every statement in the block will be terminated by a semicolon (;) which specifies compiler the end of statement.

COMMENTS in a C++ program.:

Comments are the line that compiler ignores to compile or execute. There are two types of comments in C++.

1. **Single line comment:** This type of comment deactivates only that line where comment is applied. Single line comments are applied with the help of " //" .

   e.g // cout<<tomorrow is holiday

   the above line is proceeding with // so compiler wont access this line.

2. **Multi line Comment :** This Type of comment deactivates group of lines when applied. This type of comments are applied with the help of the operators "/*" and "*/ ". These comment mark with /* and end up with */. This means every thing that falls between /*and */ is considered even though it is spread across many lines.

```
e.g   #include<iostream.h>
      int main ()
{
  cout<< "  hello world";
 /* this is the program to print hello world
      For demonstration of comments */
}
```

In the above program the statements between **/*** and ***/** will be ignored by the compiler.

CASCADING OF OPERATOR:

When shift operators (**<<** and **>>**) are used more than one time in a single statement then it is called as cascading of operators.

e.g cout<< roll<< age<< endl;

DATATYPES IN C++:

A datatype is just an interpretation applied to a string of bytes. Data in C++ are of two types:

1.Simple /Fundamental datatypes .

2.Structures/Derived datatypes.

Simple /Fundamental data types:

When programming, we store the variables in our computer's memory, but the computer has to know what kind of data we want to store in them, since it is not going to occupy the same amount of memory to store a simple number than to store a single letter or a large number, and they are not going to be interpreted the same way.

The memory in our computers is organized in bytes. A byte is the minimum amount of memory that we can manage in C++. A byte can store a relatively small amount of data: one single character or a small integer (generally an integer between 0 and 255). In addition, the computer can manipulate more complex data types that come from grouping several bytes, such as long numbers or non-integer numbers.

Next you have a summary of the basic fundamental data types in C++, as well as the range of values that can be represented with each one:

Name	Description	Size	Range
char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)

Derived Data Types:

The datatypes which are extracted / derived from fundamental data types are called derived datatypes. These datatypes can be derived by using the declaration operator or punctuators for e.g Arrays, function, Pointer, Class , Structure, union etc.

Class : A class represents a group of similar objects. To represent class in C++ it offers a user defined datatypes called CLASS .Once a Class has been defined in C++, Object belonging to that class can easily be created. A Class bears the same relationship to an object that a type does to a variable.

Syntax of CLASS:

```
Class  class_name
{
Private:
        Data members 1
```

```
                “
        Data members n
        Member functions  1
                “
        Member functions  n
Public:
        Data members 1
                “
        Data members n
        Member functions  1
                “
        Member functions  n
};//end of class
```

Class name object of Class; // creating an object of class. Private and Public are the access specifiers to the class.

STRUCTURE:

A Structure is a collection of variables of different data types referenced under one name .It also may have same data types. The access to structure variables is by default global i.e they can be accessed publicly throughout the program.
Syntax of structure.

```
 struct  structure_name
{
   Structure variable 1;
Structure variable n;
};  // end of structure
```

Structure_name structure object // creating object variable of structure.

```
 e.g
struct  student
{
    int roll;
    float marks ;
  };
```

Student s;

Access to structure variables
Structure variable can be accessed by their objects only as shown below

structure object_name. variable

e.g

student . roll

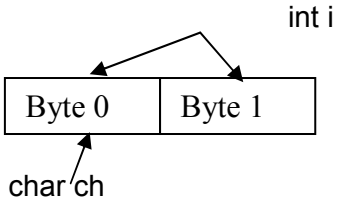
here student is the structure and roll is the member of the structure.

UNION :

A memory location shared between two different variables of different datatypes at different times is know as Union. Defining union is similar as defining the structure.

Syntax of Union :

```
   union  show
{
        int I;
        char ch;
};
```



Union show obj;

References:

A reference is an alternative name for an object. A reference variable provides an alias for a previously defined variable. A reference declaration consists of base type , an & (ampersand), a reference variable name equated to a variable name .the general syntax form of declaring a reference variable is as follows.

Type & ref_variable = variable_name;

Where is type is any valid C++ datatype, ref_variable is the name of reference variable that will point to variable denoted by variable_name.

e.g int a= 10;
int &b= a;

then the values of **a is 10** and the value of **b is also 10;**

Constant :

The keyword const can be added to the declaration of an object to make that object constant rather than a variable. Thus the value named constant can not be altered during the program run.

Syntax:-

const type name=value;

Example:-

const int **uage=50;** // it declares a constant named as **uage** of type integer that holds value 50.

Preprocessor Directives:

#include is the preprocessor directive used in C++ programs. This statement tells the compiler to include the specified file into the program. This line is compiled by the processor before the compilation of the program.

e.g #include<iostream.h>

the above line include the header file ***iostream*** into the program for the smooth running of the program.

Compilation and Linking

Compilation refers to the processing of source code files (.c, .cc, or .cpp) and the creation of an 'object' file. This step doesn't create anything the user can actually run. Instead, the compiler merely produces the machine language instructions that correspond to the source code file that was compiled. For instance, if you compile (but don't link) three separate files, you will have three object files created as output, each with the name <filename>.o or

<filename>.obj (the extension will depend on your compiler). Each of these files contains a translation of your source code file into a machine language file -- but you can't run them yet! You need to turn them into executables your operating system can use. That's where the linker comes in.

Linking refers to the creation of a single executable file from multiple object files. In this step, it is common that the linker will complain about undefined functions (commonly, main itself). During compilation, if the compiler could not find the definition for a particular function, it would just assume that the function was defined in another file. If this isn't the case, there's no way the compiler would know -- it doesn't look at the contents of more than one file at a time. The linker, on the other hand, may look at multiple files and try to find references for the functions that weren't mentioned.

ERRORS:

There are many types of error that are encountered during the program run. following are some of them:

1. **Compiler error**.: The errors encountered during the compilation process are called Compiler error. Compiler error are of two types

- Syntax error.
- Semantic error.

Syntax Error: Syntax error is the one which appears when we commit any grammatical mistakes. These are the common error and can be easily corrected. These are produced when we translate the source code from high level language to machine language.

e.g **cot<<endl;** This line will produce a syntax error as there is a grammatical mistake in the word **cout**

Semantic error: These errors appear when the statement written has no meaning.

e.g **a + b =c** ; this will result a semantically error as an expression should come on the right hand side of and assignment statement.

2. **Linker Errors.** Errors appear during linking process e.g if the word **main** written as **mian** . The program will compile correctly but when link it the linking window will display errors instead of success.

3. **Run Time error:** An abnormal program termination during execution is known as Run time Error.

e.g. If we are writing a statement X = (A + B) /C ;

the above statement is grammatically correct and also produces correct result. But what happen if we gave value 0 to the variable c, this statement will attempt a division by 0 which will result in illegal program termination. Error will not be found until the program will be executed because of that it is termed as run time error.

3. Logical Error.: A logical error is simply an incorrect translation of either the problem statement or the algorithm.

e.g : root1 = -b + sqrt(b * b -4*a*c) / (2 *a)

the above statement is syntactically correct but will not produce the correct answer because the division have a higher priority than the addition, so in the above statement division is performed first, then addition is performed but in actual practice to do addition performed then divide the resultant value by (2* a).

Manipulators :

Manipulators are the operators used with the insertion operator << to modify or manipulate the way data is displayed. There are two types of manipulators **endl** and **setw.**

1. **The endl manipulator :** The endl manipulator outputs new line . It takes the compiler to end the line of display.

```
cout << " Kendriya Vidyalaya Sangathan"<<endl;
cout<< " Human Resource and Development";
```

The output of the above code will be

Kendriya Vidyalaya Sangathan
Human Resource and development

2. **The Setw Manipulator** : The **setw** manipulator causes the number (or string) that follows it in the stream to be printed within a field **n** characters wide where n is the arguments to **setw (n).**

Increment and Decrement Operators in C++:

The increase operator (++) and the decrease operator (--) increase or reduce by one the value stored in a variable. They are equivalent to +=1 and to -=1, respectively. Thus:

C++

C +=1;

C=C+1;

are all equivalent in its functionality: the three of them increase by one the value of C.

A characteristic of this operator is that it can be used both as a prefix and as a suffix. That means that it can be written either before the variable identifier (++a) or after it (a++). Although in simple expressions like a++ or ++both have exactly the same meaning, in other expressions in which the result of the increase or decrease operation is evaluated as a value in an outer expression they may have an important difference in their meaning:

In the case that the increase operator is used as a prefix (++a) the value is increased **before** the result of the expression is evaluated and therefore the increased value is considered in the outer expression;

Example 1

```
B=3;
A =++B;          // (here A contains 4, B contains 4).
```

In case that it is used as a suffix (a++) the value stored in **a** is increased **after** being evaluated and therefore the value stored before the increase operation is evaluated in the outer expression.

Example 2

```
B=3;

A=B++;                    // (here a contains 3, B contains 4).
```

In Example 1, B is increased before its value is copied to A. While in Example 2, the value of B is copied to A and then B is increased.

Practice Session:

1. What is the name of the function that should be present in all c++ program?

Ans:main()

2. What are C++ comments?
Ans: comments are internal documentation of a program which helps the program for many purposes.

3. What is indentation of a program?
Ans: It is the systematic way of writing the program which makes it very clear and readable.

4. What is #include directives?
Ans :it instructs the compiler to include the contents of the file enclosed within the brackets into the source file.

5. What is role of main() in c++ program?
Ans:This is the first line that a C++ compiler executes. Program starts and end in this function.

6. What is a header file?
Ans:Header file provide the declaration and prototypes for various token in a program.

7. What is the purpose of comments and indentation?
Ans: the Main purpose of comments and indentation is to make program more readable and understandable.

8. What are console input /output functions?
Ans: Console I/O functions are cout and cin.

9. Write an appropriate statement for each of the following:
    1. Write the values for a&b in one unseperated by blanks and value of after two blanks lines.

    Ans: cout<<a<<b<<endl<<endl<<c;

    2. Read the values for a,b and c.

    Ans: cin>>a>>b>>c;

3. Write the values for a and b in one line, followed by value of c after two balnk lines.
Ans: cout<a<<b<<'\n\n'<<c;

10.What type of errors occurs while programming?
Ans: There are three types of errors generally occur are:
1.Syntax error
2.Semantic error
3.Type error.

11. How '/' operator is different from '%' operator?
Ans: '/' operator is used to find the quotient whereas % operator is used to find the remainder.
12. Which type of operator is used to compare the values of operands?
Ans: Relational operators.
13. How will you alter the order of evaluation of operator?
Ans: We can use parentheses to alter the order of evaluation of an equation.
14. What is the unary operator? Write 2 unary operator .
Ans : The operator which needs only one operand is called as unary operator .The '++' (increment) and '_ _'(decrement) operators.
15. What is output operator and input operator?
Ans: The output operator ("<<") is used to direct a value to standard output. The input operator (">>") is used to read a value from standard input.
16. What will be the output of following code:

```
void main()
{
int j=5;
cout<<++j<<j++<<j;  // in cascading processing starts from right to left
}
```

Ans. 7 5 5

17. What will be the output of following code:

```
void main()
{
int j=5;
cout<<++j + j++ +j++;  // values will be: 6 6 7 (From left to right)
}
```

Ans. 19
18. What will be the output of following code:

```
void main()
{
Int j=5, k;
k= a++ +a+ ++a;
cout<<k;
}
```

Ans. 18 (Because in evaluation of expression first of all prefix are evaluated, then it's value is assigned to all occurrences of variable)

Unit-3

PROGRAMMING METHODOLOGY

Algorithms and Flowcharts

After studying this lesson, the students will be able to

- *understand the need of Algorithm and Flowcharts;*
- *solve problems by using algorithms and flowcharts;*
- *get clear idea about sequential, selection and iteration construct; and*
- *understand the finite- and infinite- loop.*

Introduction

Algorithm is a step-by-step process of solving a well-defined computational problem. In practice, in order to solve any complex real life problems, first we have to define the problem and then, design algorithm to solve it. Writing and executing a simple program may be easy; however, for executing a bigger one, each part of the program must be well organized. In short, algorithms are used to simplify the program implementation. The next step is making the flowchart. It is a type of diagram that represents an algorithm or process, showing the steps as 'boxes' of various kinds and their order by connecting them with arrows. Then, the flowchart will be converted into program code.

Algorithm

An algorithm is an effective method expressed as a finite list of well defined instructions for calculating a function, starting from an initial state and initial input. The instructions describe a computation, which will eventually produce output, when executed. We can use algorithm to solve any kind of problems. However, before writing a program, we need to write the steps to solve the problem in simple English language. This step-by-step procedure to solve the problem is called algorithm.

Example

Let us take one simple day-to-day example by writing algorithm for making 'Maggi Noodles' as a food.

Step 1: Start
Step 2: Take pan with water
Step 3: Put pan on the burner
Step 4: Switch on the gas/burner
Step 5: Put magi and masala
Step 6: Give two minutes to boil
Step 7: Take off the pan
Step 8: Take out the magi with the help of fork/spoon
Step 9: Put the maggi on the plate and serve it
Step 10: Stop.

Further, the way of execution of the program shall be categorized into three ways: (i) sequence statements; (ii) selection statements; and (iii) iteration or looping statements. This is also called as 'control structure'.

Sequence statements: In this program**,** all the instructions are executed one after another.

Example

Write an algorithm to find area of a rectangle.

Step 1: Start
Step 2: Take length and breadth and store them as L and B?
Step 3: Multiply by L and B and store it in area
Step 4: Print area
Step 5: Stop

In the above mentioned example all the instructions are executed one after another. These examples are executed under sequential statement.

Selective Statements: In this program, some portion of the program is executed based upon the conditional test. If the conditional test is true, compiler will execute some part of the program, otherwise it will execute the other part of the program.

Example

Write an algorithm to check whether he is eligible to vote? (more than or equal to 18 years old).

Step 1: Start
Step 2: Take age and store it in age
Step 3: Check age value, if age >= 18 then go to step 4 else step 5
Step 4: Print "Eligible to vote" and go to step 6
Step 5: Print "Not eligible to vote"
Step 6: Stop

Example

Write an algorithm to check whether given number is +ve, -ve or zero.

Step 1: Start
Step 2: Take any number and store it in n.
Step 3: Check n value, if n > 0 then go to step 5 else go to step 4
Step 4: Check n value, if n < 0 then go to step 6 else go to step 7
Step 5: Print "Given number is +ve" and go to step 8
Step 6: Print "Given number is -ve" and go to step 8
Step 7: Print "Given number is zero"
Step 8: Stop

In the above mentioned examples all the statements are not executed, but based upon the input, some portions of the algorithm are executed, because we have 'true' or 'false' situation in the program.

Iterative statements: In some programs, certain set of statements are executed again and again based upon conditional test. i.e. executed more than one time. This type of execution is called 'looping or iteration'.

Example

Write an algorithm to print all natural numbers up to 'n'.

Step 1: Start
Step 2: Take any number and store it in n.
Step 3: Store 1 in I
Step 4: Check I value, if I<=n then go to step 5 else go to step 8
Step 5: Print I
Step 6: Increment I value by 1
Step 5: Go to step 4

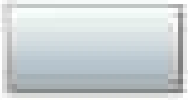
Step 8: Stop

In the above example, steps 4, 5, 6 and 7 are executed more than one time.

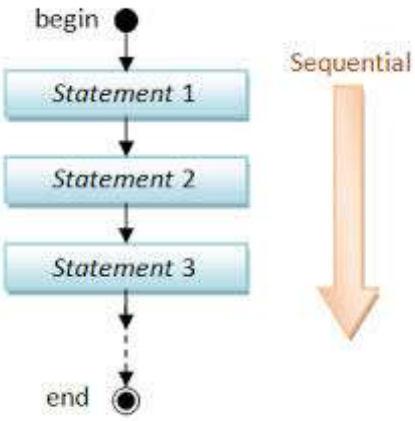
Flowchart

In the previous section of this chapter, we have learnt to write algorithms, i.e. step-by step process of solving a problem. We can also show these steps in graphical form by using some symbols. This is called flowcharting.

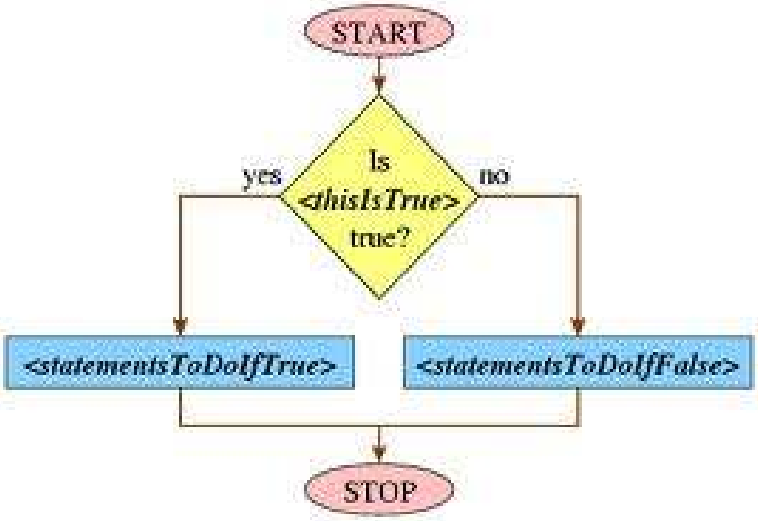
Flowchart Symbols

Symbol	Name	Function
	Start/end	An oval represents a start or end point.
	Arrows	A line is a connector that shows relationships between the representative shapes.
	Input/Output	A parallelogram represents input or ouptut.
	Process	A rectangle represents a process.
	Decision	A diamond indicates a decision.

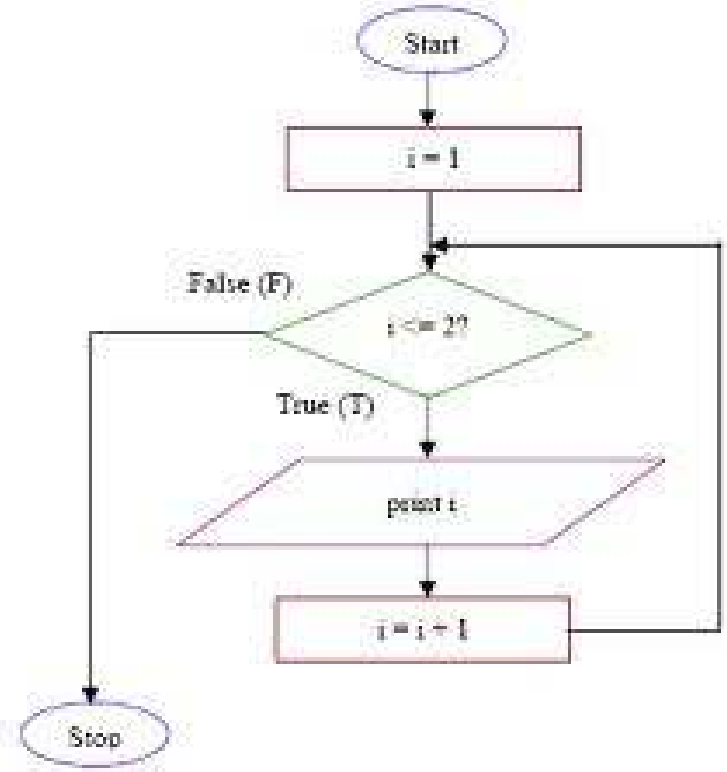
The following flowchart is an example of a sequential execution



The following flowchart is an example of a selective execution.



The following is an example of an iterative execution.



Programming Methodology

After studying this lesson, the students will be able to

- *understand the need for good programs;*
- *understand how to solve problems using different ways;*
- *get clear idea about problem solving methodology; and*
- *understand the types of errors normally occur while writing programs.*

Introduction

Learning to write computer program is very much like learning any skill. First, we should understand the problems well and then try to solve it in a logical manner. For example: We have read many books available in the market for describing the car driving methods. However, we can learn driving once we actually get into the car and start driving it. The same logic is applied in computer programming also. Computer programming is the process of writing, testing, troubleshooting, debugging and maintaining of a computer program. An effective program is that which gives result of all different inputs, including wrong input also. While creating program, we need to follow certain systematic approach. This systematic approach comprises two steps/things, viz., program structure and program representation. The program structure is implemented by using top-down or bottom-up approach and is known as 'popular approach', while the program representation plays an important role in making the program more readable and understandable.

What is a Good Program?

A Good Program means that it should produce correct and faster results, taking into account all the memory constraints. While making good program, we need to follow certain guidelines of programming language for creating a successful program. The following is the list of good programming habits that most people agree.

Clarity and Simplicity of Expression

Expressions are used to implement a particular task. It is a combination of Operators, Operands and Constants. Any expression used in the program should be understood by the user. The followings are some of the points to be kept in mind while using expressions in a program.

(i) Use library functions to make programs more powerful

Example

To find output = x_6

Output = X *X * X * X * X * X

We can use output = power (X, 6)

(ii) Follow simplicity to maintain the clarity of expression

Example

$$X = \frac{A+B}{A-B} - \frac{U+VY}{X+Y}$$

Then, we can write

X1 = (A+B) / (A-B)

X2 = (U+V*Y) / (X +Y)

X = X1 –X2

(iii) Avoid program tricks usage, whose meaning is difficult to understand by the user.

Use of proper names for identifiers

Identifiers are user defined names**.** They are used to name things. A name is associated with a function or data object (constants and variables) and used to refer to that function or data object. Identifiers are made up of letters (A-Z, a-z), digits (0-9), and the underscore character (_). They, however, must begin with a letter or underscore and not with a digit.

(i) Give meaningful name for variable (data – object) and function.

Example

To calculate Area of a Square
We use the variable names are Area and Side
Area = Side * Side.
(ii) Use proper names for constants.

Example

¶ = 3.14
Give Pi = 3.14
(iii) Do not use same name like custom, customer or account, accountant.
(iv) Do not use one letter identifiers.

Comments

A comment is a programming language construct, which is used to embed programmer-readable annotations in the source code of a computer program. Those annotations are potentially significant to programmers but typically ignorable to compilers and interpreters. Comments are usually added with the purpose of making the source code easy to understand. Hence, add comments to your code in simple English language that describes the function of the code and the reason for your decision to do it in a particular way as well.

Characteristics of good programming

Every computer needs proper instruction set (programs) to perform the required/assigned task. The quality of the program depends upon the instructions given to it. However, it is required to feed/provide the proper and correct instructions to the computer in order to yield/provide a correct and desired output. Hence, a program should be developed to ensure proper functionality of the computer and also should be easy to understand. A computer program should have some important characteristics, which are as follows:

Flexibility

A program should be flexible enough to handle most of the changes without having to rewrite the entire program. A flexible program is used to serve many purposes. For example, CAD (Computer Aided Design) software is used for different purposes such as; engineering drafting, printing circuit board layout and design, architectural design, technical drawing, industrial art, etc. Most of the programs are being developed for certain period and they need updation during the course of time.

User Friendly

A program that can be easily understood by a beginner is called 'user friendly'. It must interact with user through understandable messages. In addition, the proper message for the user to input data and to display the result, besides making the program easily understandable and modifiable.

Portability

Portability refers to the ability of an application to run on different platforms (operating systems) with or without minimal changes. Since the change of platform is a common phenomenon nowadays, due to the developments in hardware and the software, portability has to be taken care of it. In case, a program is developed for a particular platform, it would become obsolete after a certain period of time. At the same time, if a program that is developed does have the ability to work on different platforms, it makes software more useable. High language programs are often more portable than assembly language programs.

Reliability

It is the ability of a program to do its intended function accurately even if there are even small changes in the computer system. Moreover, the program must be able to handle unexpected situation like wrong input or no input. The programs, which save such ability are known as 'reliable'. For example, if the user does/gives wrong information to input, it should display a proper error message.

Self-Documenting Code

The source code, which uses suitable name for the identifiers (variables and methods), is called self-documenting code. Also, giving proper name for variables and methods would tell the reader of your code clearly -what is it doing? Hence, a good program must have a self-documenting code.

Problem solving process

The problem solving process starts with the problem specifications and ends with a concrete (and correct) program. Programming means a problem solving activity, which consists of four steps. They are;

(i) Understanding the problem;
(ii) Devising a plan;
(iii) Executing the plan; and
(iv) Evaluation

Understanding the problem

The first step is to understand the problem well. It may be very difficult to understand the problem but it is crucial. In general, one must find out the output from the given data (input data) and assess the relationship between input and output data. It is also important to verify whether the given information is sufficient to solve the problem or not.

Devising a plan

It means drawing an action plan to solve the problem, once understood. A plan is devised from data processing to the result according to the relationship that links both of them. If the problem is trivial, this step will not require much thinking.

Executing the plan

Once the plan is defined, it should follow the plan of action completely and each element of the plan should be checked as it is applied. In the course of execution, if any part of the plan is found to be unsatisfactory, the plan should be revised.

Evaluation

Finally, the result should be examined in order to make sure that it is valid and that the problem has been solved completely.

Problem solving methodology

As we all know, there are many methods/approaches available to solve a particular problem. However, the efficient way is to adopt a systematic method of problem solving. The use of systematic method of problem solving is crucial when we use a computer to solve a problem. We introduce here a seven steps problem solving method, which is closely related to the ***software life cycle*** (the various stages in the life of a program), that can be adapted by each person to solve the problem in their own style. They are given as under:

1. Problem Definition
2. Problem Analysis
3. Design the problem
4. Coding
5. Program Testing and Debugging
6. Documentation
7. Program Maintenance

Problem Definition/Specification (Theme)

Computer programs are written to solve problems posed by humankind. Prior to writing a program, one has to understand a description of the problem to solve. This description may be very precise or vague, but nevertheless, it is necessary/present.

For instance, if you want to write a program to "Find the average of five numbers",
you should ask yourself:
"What does average mean exactly?"
"How to calculate average value?"

Posing such questions compels you to define the problem very precisely. Once you are sure of what the problem entails, you must write down a list of specifications. *Specifications* are precise definitions of what the program must do. It must include the following at least:

- **Input:** what data must be included as input and in which form?
- **Output:** what data must the program produce and in which form? (in order to solve the problem)

Note: At the end of the problem definition step, you should have a list of specifications.

Problem Analysis

In this step, the problem has to be fragmented into smaller and manageable parts. The original problem has to be analyzed and divided into a number of sub-problems as these sub-problems are easier to solve and their solutions would become the components of the final program. Each sub-problem is divided into further smaller ones and this fragmentation has to be continued to achieve simple solutions. The use of modular programming is to get proper solution.

Modular Programming: Modular Programming is the act of designing and writing programs as functions (a large program is divided into the small individual components) that each one performs, a single well-defined function, which has minimal interaction between the sub-programs. It means that

the content of each function is cohesive and there is low coupling between them. There are two methods available for modular programming. They are: top-down design and bottom-up design.

Top-Down design: The principles of top-down design dictate that a program should be divided into a main module and its related module. Each module should also be divided into sub modules according to software engineering and programming style. The division continues till the module consists only of an elementary process that is intrinsically understood and cannot be further sub-divided.

Bottom-up design: Bottom-up design is just the opposite of top-down design. It refers to a style of programming, in which, an application is constructed with existing primitives of the programming language and then gradually more and more complicated features are added till applications are written. In other words, initiating the design with simple modules and then build them into more complex structures ending at the top is bottom-up design.

Designing the problem

Designing the problem can be expressed in the form of

- Algorithm
- Flowchart

Algorithm: An algorithm is a set of instructions that describe a method for solving a problem. It is normally given in mix of computer code and English language. This is often called 'pseudo-code'.

Flowchart: The algorithm is represented in the form of a diagram with action boxes linked by lines showing the order in which they are executed. This is known as 'the flow of control'. It is the diagrammatic representation of an algorithm.

Coding

The process of translating the algorithm into syntax of a given language is known as 'Coding'. Since algorithm cannot be executed directly by the computer, it has to be translated into a *programming language.*

Program Testing and Debugging

Program Testing means running the program, executing all its instructions/ functions and testing the logic by entering sample data in order to check the output. Debugging is the process of finding and correcting the errors in the program code.

Type of errors: There are three types of errors generally occur during compilation and running a program. They are (i) Syntax error; (ii) Logical error; and (iii) Runtime error.

Syntax error: Every programming language has its own rules and regulations (syntax). If we overcome the particular language rules and regulations, the syntax error will
appear (i.e. an error of language resulting from code that does not conform to the syntax of the programming language). It can be recognized during compilation time.

Logical error: Programmer makes errors while writing program that is called 'logical error'. It is an error in a program's source code that results in incorrect or unexpected result. It is a type of runtime error that may simply produce the wrong output or may cause a program to crash while running. The logical error might only be noticed during runtime, because it is often hidden in the source code and are typically harder to find and debug.

Runtime error: A runtime error is an error that causes abnormal termination of program during running time. In general, the dividend is not a constant but might be a number typed by you at runtime. In this case, division by zero is illogical. Computers check for a "division by zero" error during program execution, so that you can get a "division by zero" error message at runtime, which will stop your program abnormally. This type of error is called runtime error.

Documentation

The documentation includes the problem definition, design documents, a description of the test perform, a history of the program development and its different versions and a user's manual. Such a manual is designed for a naive user and illustrates the preparation of input data, running the program and obtaining & interpreting the results.

Program maintenance

It is not directly part of the original implementation process, but needs special emphasis. All activities that occur after a program operation are part of the program maintenance. Many large programs have long life span that often exceed the lifetime of the hardware they run on. Usually, the expenditure for the program maintenance will be more than the developmental cost of the program. The program maintenance includes the following:

- Finding and eliminating previously undetected program errors;
- Modifying the current program, often to improve its performance, or to adapt to new laws or government regulations, or to adapt to a new hardware, or to a new operating system;
- Adding new features or a better user interface, or new capabilities to the program; and
- Updating the documentation.

Maintenance is an important part of the life cycle of a program. It is also important as far as documentation is concerned, since any change pertaining to a program will require updating of internal as well as external documentation. Maintenance documentation will include results of the program development steps, design documents, program code and test information.

UNIT-4

Programming in C++

Chapter -1

Investigation of Programming Construct in C++.

OBJECTIVES :

- **to emphasize the programming construct available in C++**
- **to investigate each and every programming construct emphasizing the situation where they can be used perfectly.**
- **to demonstrate few working program in C++ using the constructs.**

1.1 Why and What of Constructs in C++ programming

1.1.1 : Flow of logic

You have seen in the previous unit that a program is implementation of an algorithm (steps involved in problem solving) using a particular programming language like C , C++ , Java etc. In our case we are using C++ as a high level language (HLL) to code our programs.

A programming language like C++ is having a bundle of programming elements like tokens , identifiers , variables , constants , operator symbols , punctuation symbols etc. (please refer to earlier unit if you have not learnt all these terms.) . All these programming elements helps a programmer to write simple C++ programs as shown below :

program 1.1

```
// a simple program in C++ to add two integers
#include<iostream.h>
void main( )
{

      int a = 0 ,  b = 0 ;
      cout<<" Input two numbers" ;
      cin>> a>>b;
      cout<<" The sum of numbers are: "  << (a+b);
}
```

The above program is a very basic program which asks two integer values from user and prints out their sum. In the above program you may observe that various programming elements like:

< ( ) { = , a b << >> int void main "Input two numbers " ;
are being used.

Work out yourself :

Identify each of the above programming elements and write its name using a table

Students the program above is very simple to be thought , written and executed, but in real industrial programming situation we seldom get such programs to write. There are many complex situation in life where we often need to write programs using few advanced programming elements. Let us investigate few of the real life situations :

a) Ram needs to find out whether any number is divisible by both 3 and 5.

b) A shopkeeper wants to give x % discount on a particular purchase only when the net purchase by his customer exceeds Rs. 1000

c) A teacher wants to calculate the average marks for each of his 40 pupils in a class.

d) Suresh wants to continue his program till he is pressing escape button on keyboard.

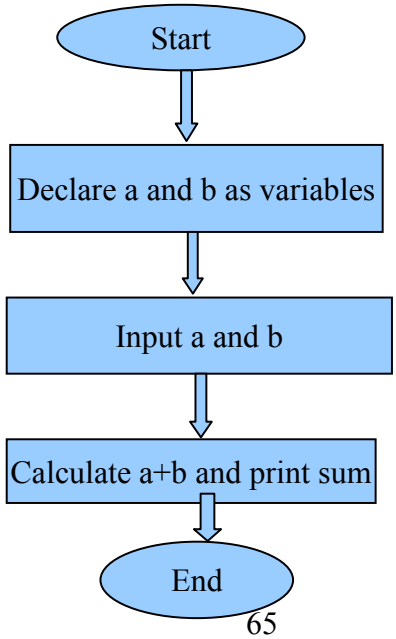
Can you program the above programming problems using the programming elements of C++ which you have learn till now ?

Your answer would be probably a NO , because the C++ programming elements which you have studied till now , will only enable you to write **simple linear programs , or sequential programs**.

1.1.2: ***Linear Flow***

A simple linear program is one which has a linear flow of execution of programming logic /statements. e.g. program 1.1. The logic of program flows (or better they are termed as control flow) from top to bottom out of set of statements.

Now let us proof the that program 1.1 follows a sequential flow of logic by the following flow chart diagram :



flowchart : 1

So easy isn't it ? As you observe that the arrows are moving in one direction only from top to bottom , the flow of program is sequential / linear.

Work out yourself :

> Identify few of the programs which you have written while dealing Unit-3 and find whether they are following sequential flow of program? If yes can you make a flow chart to justify your idea.

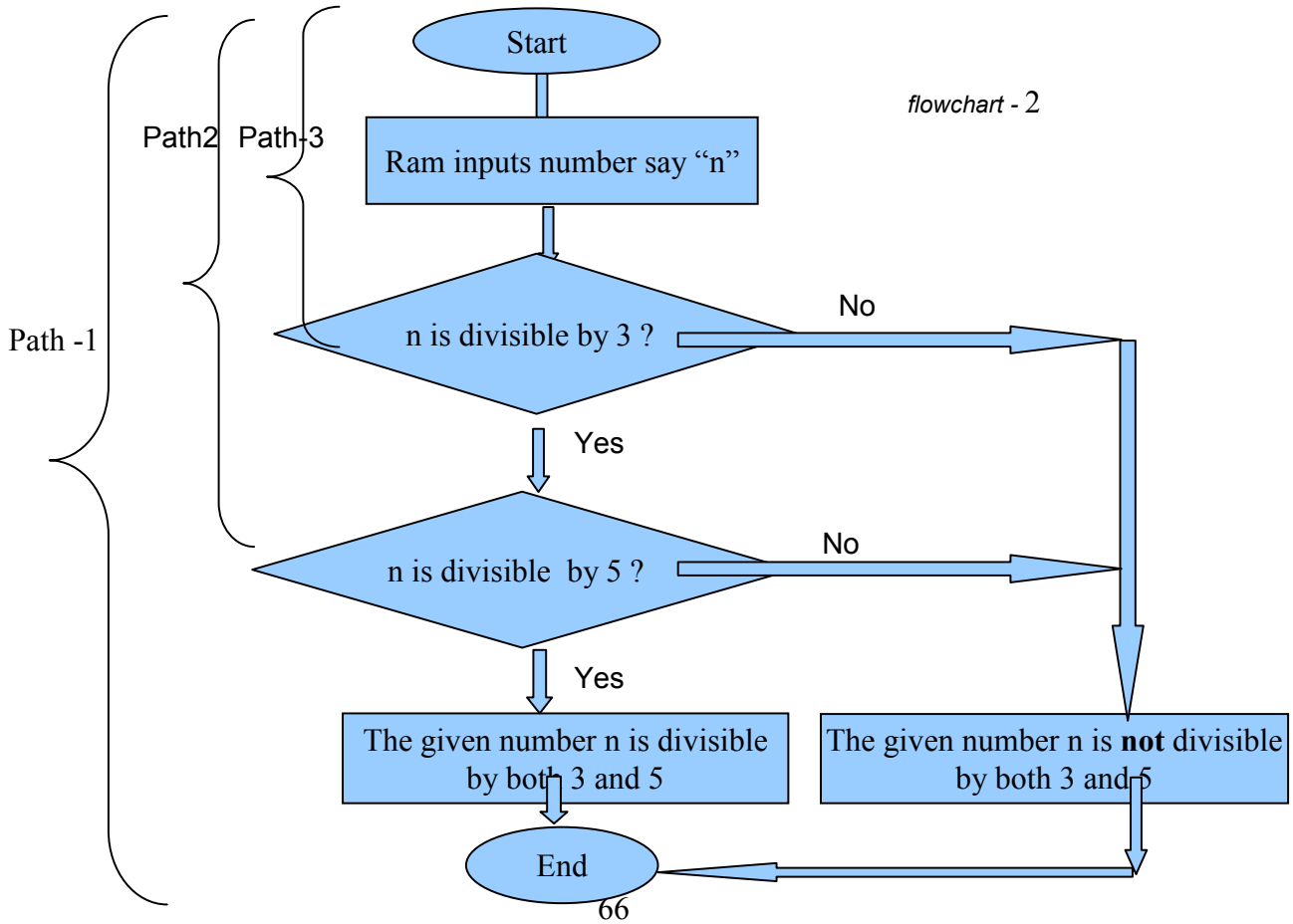
So there exist a big question now that whether C++ programs have other types of flow of program which is totally different from linear flow. Yes , students we also have other types of flow of program.

Conditional Flow

Let us investigate the real life problems visited by us earlier in this chapter :

a) Ram needs to find out whether a number is divisible by both 3 and 5

We will represent the above problem in flow chart notation :



flowchart - 2

Observe the above flowchart-2 and compare it with the earlier flowchart – 1 and then answer the following questions :

i) Are arrows in the flowchart (which represent flow) emerging and proceeding in one direction in both flowcharts ? [Yes / No].

ii) If answer to the first question is No then why not ?

iii) Let us call the flow of arrows in a particular direction a **Path** then how many such paths you observe in first flowchart and that in second one ?

iv) In flowchart – 2 we have used rhombus. What is its significance in whole flowchart?

So going through the questions above and answering them yourself you find that in the second flowchart is different from the first one in following respect :

1) The direction of arrows in second flowchart is not always linear, sometime the direction of the flow takes turns at the junction of a rhombus. (there are two such such turnings)

2) At each rhombus junction the flow decides which way to proceed after asking a logical statement (e.g n is divisible by 3) , thus creating several paths following which the program logic can flow. In our flowchart -1 there is no such branching of paths is observed, but in case of flowchart-2 we may observe two branching where from a program can flow through to reach end and terminate itself.

So we may now emphasize that a program not only have a linear flow but it can also have a branched flow as observed in flowchart- 2. Each of these branches can be interpreted as one execution path of the same program. Hence we can say that in real world, there exist some programming situation which needs branched flow of program instructions, or in simple terms we may say that the program demands several paths of execution.

Thus,

A program having multiple paths of execution, where each path leads to different type of completion states, are categorized as Conditional Programs. The selection of paths of execution depends upon a logical decision being made (look at the rhombuses)

When we consider our problem then it is seen that a number when it is satisfying both the logical conditions at two rhombuses ends up with a result that it is divisible by both values 3 and 5(execution path -1) whereas if it not satisfying either of the two logical conditions placed at two rhombuses then the program ends up with another result showing it is not divisible by both of them together. Satisfying any one of the logical condition would not solve our purpose as we are expecting a logical AND (&&).

Now let us convert the above flowchart -2 in a C++ program. You may not understand all the parts of programs here, so don't worry it will be described in the later part of the chapter.

Program – 1.2

```
// program to implement the problem described by flowchart-2

#include<iostream.h>
void main( )
{
        int num = 0 ;    // declaring a variable num

        cout<<"Input the number you want to check" ;
        cin>>num;                // Inputting value of n

    if ( n % 3 == 0 )   // checking whether n is divisible by 3 , consider the first rhombus of
    {                           the flowchart-2
        if (n% 5 == 0 ) // checking whether n is divisible by 5 , consider the second rhombus
        {                       // of flowchart-2

             cout<<"The number you inputted is divisible by both 3 and 5" ;    // path1
        }
        else
            cout<<"The number you inputted is not divisible both by 3 and 5";   // path2
    }
    else
      cout<<"The number you inputted is not divisible both by 3 and 5";   // path2

}
```

The shaded part of the above program shows how a conditional branching is implemented with an if ( )- else construct of C++.

Workout yourself :

Draw flowcharts for problem situation b) given earlier in page no. 4, and then find whether the problem follows a linear flow or conditional flow ? Check it out with you teacher.

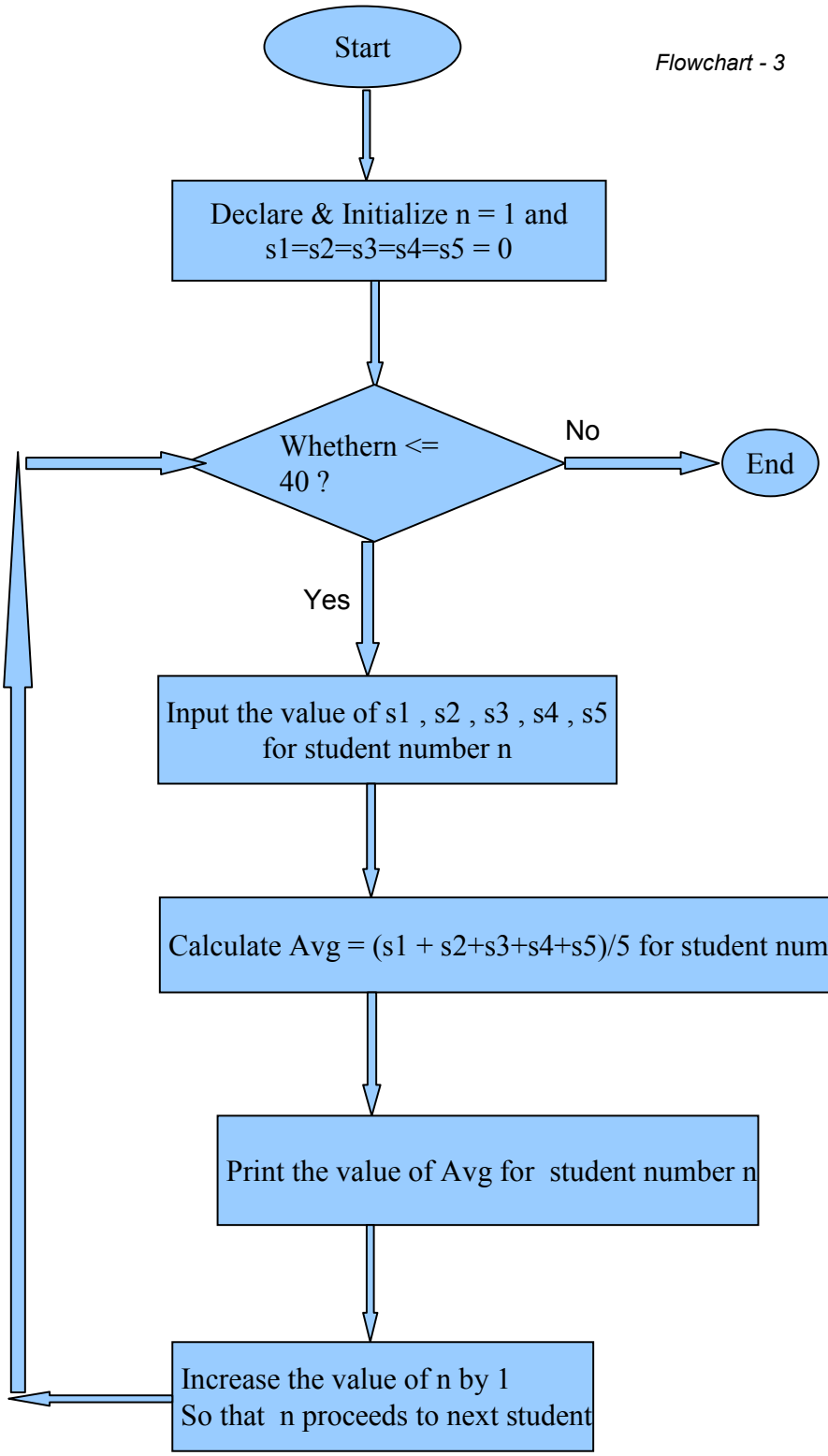
1.1.3 : Iterative Flow / Cyclic Flow :

Now let us solve the problem situation c) given in page 4 with the help of a flowchart . i.e

A teacher wants to calculate the average marks for each of his 40 pupils in a class.

Let us assume that the teacher maintains count of his/her students in a variable say n.
Let the teacher starts n from 1st student i.e n = 1
Let there are marks in five subjects say s1 , s2 , s3 , s4 and s5 and Avg as Average

We must remember that this problem demands calculation of average marks of each student, i.e Average marks for each of the individual 40 students would be calculated and displayed.


Start
Flowchart - 3
Declare & Initialize n = 1 and
s1=s2=s3=s4=s5 = 0
Whethern <=
40 ?
No
End
Yes
Input the value of s1 , s2 , s3 , s4 , s5
for student number n
Calculate Avg = (s1 + s2+s3+s4+s5)/5 for student number n
Print the value of Avg for student number n
Increase the value of n by 1
So that n proceeds to next student

Observe the above flowchart-3 and compare it with flowchart-2 given earlier. Try to answer the following question :

a) Are directions of arrows in both flowchart 2 and 3 are always pointing downwards ? **[Yes / No]**

b) If your answer to the above question a) is in No then what are other directions in which arrows proceeds? **[choose from : Leftwards , Rightwards , Upwards , all of them]**

c) In which flowchart the arrow proceeded upwards and Why ?

Verify your answer with your teacher.

Sometime according to special problem situation we find that in a program we proceed to a same statement / step again , which we have visited earlier during flow of program. Observing flowchart – 2 we find that we never proceed to a step twice.

In case of flowchart-3 we observe that steps after the rhombus are repeated if logic in rhombus produces a Yes value (you may see that an arrow moves upwards to the rhombus). **A cycle is formed between the Rhombus and steps after it, based on the Yes value of the logic (n <= 40 ?) kept in rhombus.**

This cyclic path of execution is referred as an **Iteration** in C++ programming jargon.

Workout yourself :

i) How many times the program proceeds in a Cyclic path in the flowchart- 3. Why ? When does this cyclic path of execution ends in the flowchart-3.

ii) What will happen if the cyclic path of execution never gets ended?

The cyclic path of execution or Iteration in problem represented by flowchart-3 ends **based on the No value of the logic (n <= 40 ?) kept in rhombus.** This **No** will be generated when n will become anything greater than 40.

Now let us convert the above flowchart -3 in a C++ program. You may not understand all the parts of programs here, so don't worry it will be described in the later part of the chapter.

Program – 1.3

```
// program to implement the problem described by flowchart-2



#include<iostream.h>
void main( )
{
        int n = 1  ;                    // declaring a variable n to keep count of students
        int s1 = s2=s3=s4=s5=0;         // declaring subject marks and initializing them with 0
        float Avg = 0.0 ;               // declaring variable for keeping Average of student's
marks
        cout<<"Input the number you want to check" ;
        cin>>num;               // Inputting value of n
```

```
    while ( n <= 40 )                  // Iteration starts here
    {
        cout<<"Input values for s1 , s2 , s3 , s4 and s5" ;    // path1
        cin>>s1>>s2>>s3>>s4>>s5 ;
        Avg = (s1+ s2 + s3 + s4 + s5) / 5 ;


        cout<<"\nThe average marks scored by student no." <<n << " is "<<Avg ;

        n++ ;    // Incrementing the value of n by 1
    }

    // when cyclic condition is not satisfied the program will flow out here and terminate.
}
```

The shaded part of the above program shows how an Iteration is implemented with a **while( )** construct of C++. A while( ) construct instructs the compiler to repeat the set of statement under its scope { } if the logical expression (here : n <= 40) attached with it produces a True value. If the logical statement fails to produce a True value (i.e. when n becomes 41) then the iteration is ended just by terminating the scope of while( ) construct and thereby terminating the program.

The Iterative flow of program is also called as Looping , in above program we have used a while( ) loop construct.

Let us summarize what we have learned till now :

1. A program flow is the direction following which steps of program gets executed.
2. A program can have three types of flow of logic :
   - Sequential Flow // refer to flowchart-1 and program 1.1
   - Conditional Flow // refer to flowchart-2 and program 1.2
   - Iterative Flow // refer to flowchart-3 and program 1.3
3. The choice of flow of program is decided by a programmer based on the problem situation. Hence you as a student must analyze the problem very well before before categorizing the problem as sequential , conditional or Iterative program.
4. A programming construct in C++ is just a keyword which governs the flow of logic / flow of control in a program and decides the various paths which a program may follow during its lifetime till it ends.
5. An if ( )- else construct governs the conditional flow of logic, whereas a while ( ) construct governs the Iterative flow of logic or a loop.

Workout yourself :

Draw flowcharts for problem situation d) given earlier in page no. 4, and then find whether the problem follows a linear flow or conditional flow ? Check it out with you teacher.

Check your progress:

Here are few programming situations given to you find categorize each of them according to the type of flow of control they require , i.e. Sequential , Conditional or Iterative by writing S , C and I before them :

1. Dinesh wants to find simple interest on a given amount at a particular rate of interest for fixed number of years.

2. Adarsh wants to compute compound interest on a given principal , rate and time but without using compound interest formula taught to him in class VIII.

3. Mahalaxmi wants to check whether her weight is an odd number or even.

4. Surekha wants to calculate area of a triangle using Heron's formula.

5. Ravi while designing a game program , wants that his game character kicks his motocycle 5 times before the motorcycle gets started.

6. Ayush wants to calculate factorial of a number if the number is even otherwise he wants to find its reciprocal.

7. Mera Bank wants that its customer will be able to draw money from there account only when there is a minimum balance of Rs. 1000 left in their account after the withdrawal.

Check and discuss your answers with your teacher.

Chapter -2

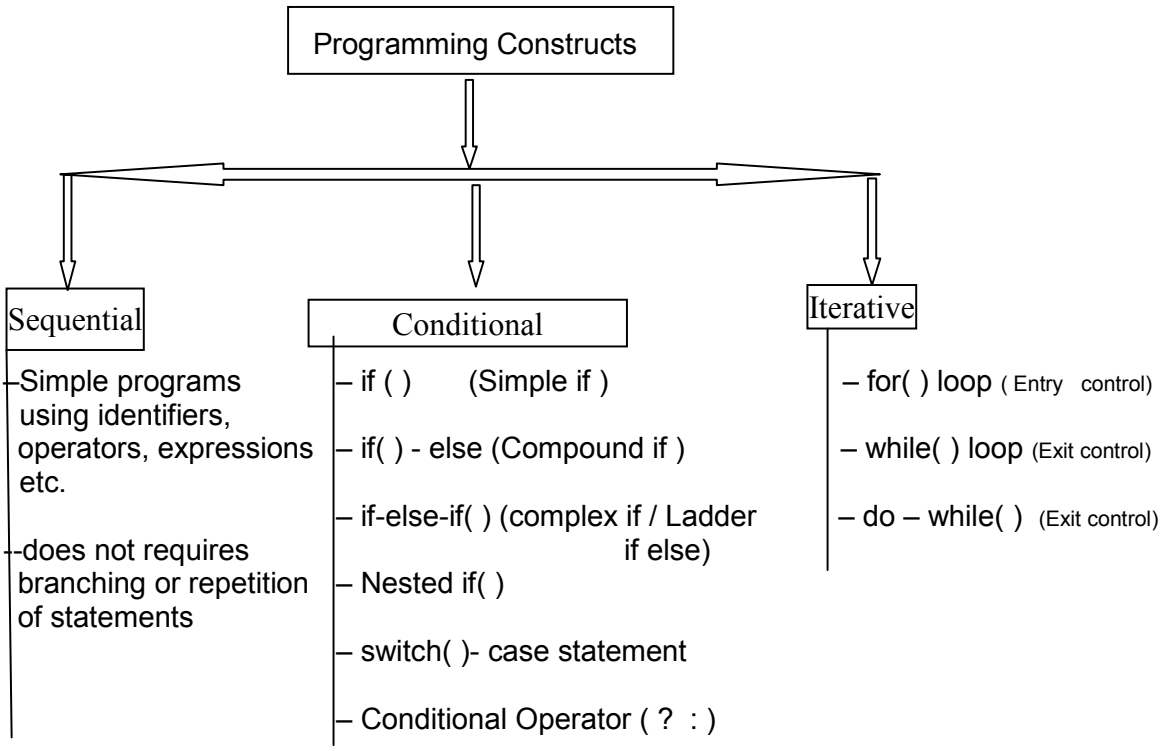
Using C++ constructs

Objectives:

- **to analyze syntaxes of various programming constructs available in C++.**
- **to draw comparison between various programming constructs.**
- **to apply the syntax of various programming constructs in problem solving**

2.1 Categories of available constructs

After exploring into various types of flow of control / logic in different programming situations let us go through the detailed syntax/format of each of the programming constructs available in C++, using which we can monitor flow of control in our program. Here is one diagram which categorizes C++ constructs in detail :


Programming Constructs
Sequential
Simple programs using identifiers, operators, expressions etc.
does not requires branching or repetition of statements
Conditional
– if ( ) (Simple if)
– if( ) - else (Compound if)
– if-else-if( ) (complex if / Ladder if else)
– Nested if( )
– switch( )- case statement
– Conditional Operator (? :)
Iterative
– for( ) loop (Entry control)
– while( ) loop (Exit control)
– do – while( ) (Exit control)

Let us deal each one of them one after another in detail.

2.1 Conditional Constructs :

2.1.1 : Simple If ( )

```
syntax :
        if ( <conditional expression / logical statement>)
        {
            // statements to be executed when logical statement is satisfied
            // i.e. when the logical statement yields a true value
        }
```

points to remember :

i) a logical statement always evaluates as true / false.
ii) any value in C++ other than zero (positive / negative) is considered to be true whereas a zero (0) is considered to be false.
iii) < > in syntax is known as a place holder, do not type it while writing program. It only signifies that any thing being kept there varies from program to program.
iv) if there exists only one line of program statement under if( ) scope then we may ommit curly braces { }

The statement kept under simple if ( ) gets executed only when the conditional expression/logical statement under it is evaluated as true.

Examples :

```
int x = 1 , y = 3;
x += y;
if ( x > y )
{
   cout<<"x is greater";
}
```

In the above example the conditional statement under if ( ) will be always evaluated as true because the value of x will become 4 before the comparison thus the expression 4 > 3 yields a true value letting the statement under if( ) to execute i.e. the output of the above code snippet would be " x is greater "

Workout yourself :

Convert the above code snippet into a program in your practical period and execute it, verify the output.
Now change the initial value of x to 0 instead of 1 i.e. Change x = 0 in the program. Execute the

2.1.2 : Compound If ( ) : if – else combination

syntax :

```
if ( < conditional statement > )
{
    // statements to be executed when logical statement is satisfied
    // i.e. when the logical statement yields a true value
}
else
{
   // statements to be executed when logical statement is not satisfied
    // i.e. when the logical statement yields a false value
}
```

Example :

```
int x = 0 , y = 3;
x += y;
if ( x > y )
{
   cout<<"x is greater";
}
else
{
   cout<<"we are in else part because x and y both became equal.";
}
```

The above code snippet (portion) has two different paths of execution. If the conditional statement under if( ) is evaluated to be true then the statement under if ( ) block will be executed otherwise the statements under else block would be executed. The above code produces an output as "we are in else part because x and y both became equal." because the conditional statement under if ( ) evaluates as false as x is not greater than y, it is same as that of y.

Workout yourself :

In the above code snippet modify the logical statement under if ( ) such that the if ( ) block gets executed instead of else( ) block. You should not modify the initial values of x and y or change number of variables in the program.

2.1.3 : Complex If ( ) : if – else ladder

syntax :

```
if ( <condition -1> )
{
   // do something if condition-1 is satisfied
}
else if ( <condition – 2 >)
{
```

```
    // do something if condition -2 is satisfied
}
else if (<condition – 3 >)
{
    // do something if condition- 3 is satisfied
}
                :
                :    // many more n-1 else - if ladder may come
                :
else if( < condition – n >)
{
        // do something if condition – n is satisfied
}
else
{
        // at last do here something when none of the above else-if( )
        //conditions gets satisfied.
}
```

In the above syntax there are ladder of multiple conditions presented by each if( ) , all of these conditions are mutually exclusive that is only one of them would get satisfied and all the conditions below it would not be evaluated and is discarded.

Say suppose if condition-3 gets satisfy i.e. it yields a true value for the condition, the statement under the block of third if( ) gets executed and all other n number of if( ) conditions below it would be discarded.

If none of the n if ( ) conditions gets satisfied then the last else part always gets executed. It is not compulsory to add an else at the last of the ladder.

Example :

```
char ch = getch( );
if ( ch >= 'a'   && ch <= 'z' )
{
  cout<<"you have inputted a lowercase alphabet";
}
else if ( ch >= 'A' && ch <= 'Z' )
{
  cout<<"you have inputted an uppercase alphabet";
}
```

```
else if ( ch > '0' && ch <= '9' )
{
  cout<<"you have inputted a digit";
}
else
{
  cout<<"you have inputted any special character or symbol";
}
```

In above code snippet a character is being inputted from user and is being checked upon by various if ( ) condition as alphabets, digit or any other special symbol. If the first condition gets satisfied then the character inputted is a lower case alphabet, if not then the second if ( ) is evaluated , if the second if( ) gets satisfied then the character is an upper case alphabet, if not then the third if ( ) is being evaluated , if it is satisfied then the character is a digit if not then finally it is inferred as any other symbol in the last else( ) .

The benefit of this type of conditioning statement is that we can have multiple conditions instead of just having one or two as seen in case of earlier if( ) constructs.

Workout yourself :

A date in the format of dd/mm/yyyy when dd, mm, and yyyy are inputted separately is said to be a valid date if it is found in a particular year's calendar. For example 22/12/2012, 29/02/2012 are valid dates in calendar of 2012, but 31/06/2012 or 30/02/2012 are invalid dates in calendar of 2012.

Find out how many ladders of if( ) conditions would be required to write a program for checking validity of a date.

Discuss your answer with your teacher.

2.1.4 : Nested if-else

You might have seen a crow's nest or any other bird's nest , the materials being used to build the nest are enclosed within one another to give ample support to the nest.

We also have the same concept of nesting of constructs within one another , to give ample strength to our program in terms of robustness , flexibility and adaptability (these terms you have learned in Unit - 3 earlier.)

We are now considering nesting of an if ( ) construct within another if ( ) construct i.e one if( ) is enclosed within the scope of another if( ). The construct thus formed is called nested if( ). Let me show you few of the syntax forms of nested if( ) :

Syntaxes: **Simple if ( ) nested within scope of another simple if ( )**

```
if ( <outer- condition > )
{
        if ( <inner-condition> )
        {

                //some statements to be executed
                // on satisfaction of  inner  if ( ) condition.

        } // end of scope of inner if( )
```

```
        //some statements to be executed
        // on satisfaction of outer  if ( ) condition.

    } // end of the scope of outer if( )
```

compound if ( ) nested within scope of another compound if ( )

```
if ( <outer- condition > )
{
        if ( <inner-condition> )
        {
                //some statements to be executed
                // on satisfaction of  inner  if ( ) condition.

        }
        else
        {
                // statements on failure of inner if( )
        }

      //some statements to be executed
      // on satisfaction of outer  if ( ) condition.

}
else
{

        // statements on failure of outer if( )
}
```

Ladder if-else-if ( ) nested within scope of another ladder if-else-if ( )

```
if ( <outer- condition-1 > )
{
        if ( <inner-condition - 1> )
        {

                //some statements to be executed
                // on satisfaction of  inner  if ( ) condition.

        }
        else if ( <inner-condition – 2> )
        {
                // statements on failure of inner if( )
        }
        else
        {
                // last statement of the inner ladder if-else-if
        }
```

```
        //some statements to be executed
        // on satisfaction of outer  if ( ) condition.

    }
    else if ( <outer-condition-2 >)
    {

            // statements on failure of outer if( )
    }
```

Like this you may try to keep any type if( ) construct within the scope of other type of if ( ) construct as desired by the flow of logic of that program.

Workout yourself:

I) Try to nest simple if ( ) upto 4 levels ii) Try to nest complex if within a compound if ( )
iii) Try to nest three simple if ( ) within another simple if( )

let us now explore few example programs using various if( ) constructs :

program 2.1

// program to compare three integer values to find highest out of them

```
#include<iostream.h>
void main( )
{
        int  n1= n2 = n3 = 0;
        cout<<"Input three integers";

        if( n1 > n2 && n1 > n3)
            cout<<n1 << " is the highest value";
        else if( n2 >n1 && n2 > n3)
            cout<< n2 << " is the highest value";
        else if ( n3 > n1 && n3 > n2 )
            cout<< n3 << "is the highest value";
        else
            cout<< "all values are equal";

}
```

Explanation :

Lets parse the logic of the above code (shaded area) to understand it using a dry run :

Case – I

let us assume that the value of **n1 = 1 , n2 = 5 and n3 = -7**

evaluation of the first if( ) in the ladder :

```
if( n1 > n2  &&  n1 > n3  )
==> if( 1 > 5   &&  1 > -7 )
==> if ( false && true )
==> if( false )   // as per truth table of logical and ( && ) operator
```

since the first condition is evaluated as false so the next condition in the ladder would be evaluated

```
if( n2 > n1 && n2 > n3 )
==> if ( 5 > 1 && 5 > -7 )
==> if ( true && true )
==> if(true)  // as per truth table of logical and ( && ) operator
```

since the second logic is evaluated as true, it open the block of second if( ) and the statements under the second else-if ( ) block gets executed and an output is printed on the console as :

"5 is highest value"

Workout yourself :

I) Try to execute the above program 2.1 using dry run method with values n1 = 1 , n2 = 1 and n3 = 1

ii) Try to implement the above program using any other type of if( ) construct

program 2.2

```
// program to find whether a 4 digit inputted year is a leap year

#include<iostream.h>
void main( )
{
        int year = 0 ;
        cout<<"Input a 4 digit year"
        cin>> year;

        if ( year % 4 == 0 )
        {
           if ( year % 100 == 0 )
           {
                 if( year % 400 == 0)
                 {
                     cout<< "Year : " << year <<" is a leap year";
                 }
                 else
                 {
                     cout<<"Year : "<< year << " is not a leap year";
                 }
            }
           else
            {
                 cout<<"Year : " << year <<" is a leap year";
             }
        }
        else
```

```
        {
                cout<<"Year : "<< year << " is not a leap year";
        }


    } // end of main ( )
```

The above program is a very good example showing the use of nesting of if( ).

Explanation:

Let us first understand which year would be called leap year , a leap is a year which :

i) is divisible by 4 but not divisible by 100
ii) is divisible by 4 as well as divisible by 100 and at the same time divisible by 400

Any other criteria will make the year as a non-leap year candidate.

Dry Run -1

Let us parse the gray area code of the above program with a dry run having year = 1994

```
if( 1994 % 4  == 0)
==> if ( 2 == 0 )
==> if ( false )
```

Since the first if condition is not satisfied the flow of the program proceeds to its else block and prints the output as " Year : 1994 is not a leap year"

Dry Run – 2 :

Let us parse the gray area code of the above program with a dry run having year = 2000

```
if( 2000 % 4 == 0)
==> if ( 0 == 0 )
==> if ( true )
```

Since the first condition evaluates out to be true it opens up its block and the next statement in the inner block gets executed as :

```
==> if (2000 % 100 == 0)
==> if ( 0 == 0 )
==> if ( true )
```

Since the first nested if ( ) condition evaluates out to be true it opens up its block and the next statement in the inner block is continues as :

```
==> if( 2000 % 400 == 0)
==> if ( 0 == 0 )
==> if ( true )
```

Since the second nested if ( ) condition evaluates out to be true it opens up its block and the next statement in the inner block is continues to print output on console as :

"Year : 2000 is a leap year"

Workout yourself:

i) Google out to find that whether year 1900 was a leap year or not and then verify it using a dry run with the help of above program.
ii) Try to implement the above code without using nested if ( ) construct.

Check your progress:

1. Find error in code below and explain.

```
#include<iostream.h>

void main()

{

        int x = 5 ;

        if( x > 5 )

        {

          x += 5;

          int y = 8;

        }

        y += x;

}
```

2. Find the output of the code below :

```
void main( )
{
        int NoOfGirls = 4;
        int NoOfBoys = 10 ;

        if ( NoOfBoys =  8    &&    NoOfGirls <= NoOfBoys )
                cout<<"Great achievement";
        else
                cout<<"Greater achievement";
}
```

3. Find the output of the code below :

```
void main( )
{
        int circle = 5 , rectangle = 0 , square = 4 , triangle = 0 ;
        if( circle)
        {
```

```
                if( rectangle || square )
                {
                        cout<<"Draw diagram";
                }
                else if( ! rectangle  && ! square )
                {
                        cout<<"Invalid diagram";
                }
                else
                {
                        if( circle == rectangle || square == triangle )
                        {
                           cout<<"Canvas Available";
                        }
                }
        }
        cout<<"Invisible diagram";
}
```

4. Find the output of code below:

```
void main( )
{
        int x = 3 , y = 5;
        if( x <= 5 );
          cout<<"Hurray";
        else
          cout<<" Trapped";
}
```

5. Write a program which inputs day(dd) , month(mm) and year(yyyy) and checks whether it is a valid date or not. [A valid date must lie on the calendar]

6. Dipu Jwellers gives discount on a fixed purchase total based on following criteria :

Offer	Offer Months	Discount %	Purchase total
Winter Bonanza	Oct to Feb	30	Between 3000 to 5000
Summer Bonanza	Mar to June	20	Between 10000 to 12000
Monsoon Bonanza	July to Sep	10	Between 2000 to 10000

If the purchase amount of a customer does not lies between the given purchase total then no discount should be given.

Write a program to calculate the Bill amount of a purchase after giving proper discount.

2.1.5: switch-case statement

switch-case construct is a type of conditional construct which resembles the same flow of logic as that of ladder if-else-if with few exceptions. Let us observe the syntax of the switch-case construct of C++ :

Syntax :

```
switch ( <switching variable | switch expression >)
{
        case <value-1> :
                //do something if switching variable value matches
                // with that of value-1
                break;
        case <value-2> :
                //do something if switching variable value matches
                // with that of value-2
                break;
        case < value-3> :


                //do something if switching variable value matches
                // with that of value-3
                break;
                        :
                        :
                        :
        case <value-n> :
                //do something if switching variable value matches
                // with that of value-n
                break;
        default :
                //do something if switching variable value does not matches
                // with any value between value-1 to value-n
}
```

A switch-case control matches a particular switching value in a switch variable or a value generated after evaluating an expression with several values, when it finds an exact match between switch variable value and case value it enters into that particular case to execute the statements under that case, and then on finding a break statement it jumps out of the case and then comes out of the scope of switch-case without considering other cases value.

Example:

```
int menu_choice = 0;
cin>>menu_choice ;
switch (menu_choice)
{
```

```
        case  1 :
           cout<< “press key 's' to start the game”;
           break;
        case  2 :
           cout<<”press key 'n' to navigate through game plan”;
           break;
        case 3 :
           cout<<”press key 'c' to change the level of the game”;
           break;
        case 4 :
           cout<<”press key 'f' to fast your speed”;
           break;
        case 5 :
           cout<<”press key 'x' to exit from game”;
           break;
        default :
           cout<<”you have to choose between 1 to 5” ;
}
```

The above code implements a game menu design where inputted integer value in the switching variable menu_choice is being matched with values 1 to 5 , each value representing one game action.

You may observe that the flow of the switch-case construct is same as ladder if-else-if construct, because it also deals with multiple paths of execution out of which a selected path gets executed. If none of the conditions gets satisfied then the statements under the default case gets executed.

Points to remember while using a switch-case construct in a program :

a) a **break statement** always ends a case, if break is not placed then all the cases after the current case gets executed till a break statement is met.

b) a switch-case always matches its switch variable value with a single constant case value, this case value must always be a single integer value or a single character. Floats and double values of switch variable are not valid. i.e. the code below is invalid code :

```
float v = 0.0
cin>> v;
switch( v)
{
    case 1.1 :
        // do something
        break;
       …
}
```

Invalid switch variable declaration

Invalid case value

c) the cases in the switch cannot provide a range of values to be matched with the

switching variable i.e. we can't write statements like >= 4 && <= 9 with cases. The following code is invalid :

```
char ch = '*' ;
cin >> ch;
switch(ch)
{
        case >='a' && <='z'

          cout<<"You have entered lowercase alphabet";
          break;
        case >= 'A' && <= 'Z'
          cout<<"You have entered uppercase alphabet";
          break;
}
```

The above code gives us an example where ladder if-else-if has an advantage or switch-case, because we can have a logical statement having a range of values to be compared.

d) The sequence of case value does not matters i.e. It is not compulsory to keep a lower case value as first case and the highest case value being the last case. They can exist in any order.

e) The default case is optional and should be always kept at the last place in switch-case construct

f) A switch-case construct can also be nested within another switch-case operator.

Workout yourself :

i) Write a program which inputs a month number from user and then finds the total days present in that month. Use only switch-case construct. For example if user inputs month number as 2 then the program displays "February has 28 or 29 days".

ii) Compare and contrast switch-case construct with ladder if-else-if construct in a tabular form.

2.1.6 : Conditional operator (< > ? <> : <>)

Conditional Operator is a small short hand operator in C++ which helps to implement flow of logic based on some condition like if-else construct. The syntax of the conditional operator is :

Syntax :

```
<logical expression> ? < true part > : < false part > ;
```

The logical expression in the operator gets evaluated either as true or false , if true then statement after symbol (?) gets executed otherwise statement after symbol (:) gets executed. It acts much like an if ( ) - else construct but can only execute single statement. Like if-else the conditional operator

cannot have a block of code to execute in its true or false part.

We often use conditional operator to implement a short one line conditional expression.

Example:

```
int x = 5 , y = 7;
int result =  ( x > y )  ? 1 : 0;
cout<< result;
```

The output of the above example would be 0 because the condition x > y is not being satisfied , hence the value 0 is being assigned to result. A conditional operator can be also nested within another conditional operator.

Workout yourself:

Write a program using conditional operator to find the highest of three inputted integers. [Hint : use nesting of conditional operator]

2.1.7 : Nesting of all conditional constructs

In real life programming situations often all the conditional constructs are being used in a single program. Any of the conditional construct can be nested within any of the other construct. For example a switch-case construct can nest a ladder if-else-if within its scope, or each ladder condition of a ladder if-else-if may nest a switch-case construct within its scope. The following code snippet justifies this idea :

```
switch( < an expression> )
{
        case <value-1> :
                if( < conndition -1 >)
                {
                  // some code
                }
                else if (< condition-2>)
                {
                   // some code
                }
                else
                {
                   // some code
                }
            break;
        case <value-2 >
                if( < condition -1 >)
                {
                  // some code
                }
                else if (< condition-2>)
                {
                   // some code
                }
```

```
                    else
                    {
                       // some code
                    }
                break;
                    …
} // end of switch-case
```

Similarly we can have :

```
if ( <condition -1 >)
{
   switch(var1)
   {
        case <value1> :
                …
                break;
        case <value2>
```

```
                …
                break;
        case<value-3>
                …
                break;
}
else if ( <condition-2>)
{
        switch(var1)
   {
        case <value1> :
                …
                break;
        case <value2>
                …
                break;
        case<value-3>
                …
                break;
     }
}
...
```

Ask your teacher which of the constructs can be nested within which of the other constructs.

Check Your Progress :

1. Find the output of the code given below :

```
#include<iostream.h>
void main( )
{
        int x = 3;
        switch(x)
        {
                case 1 :
                  cout<< “One”;
                  break;
                case 2 :
                  cout<< “Two”;
                  break;
                case 3 :
                  cout<< “Three”
                case 4 :
                  cout<< “Four”;
                case 5:
                  cout<< “Five”;
                   break;
        }
}
```

2. Find error in following code :

```
#include<isotream.h>
void main()
{
  int a = 10;
  int b = 10;
  int c = 20;

 switch ( a ) {
        case b:
                cout<<”Hurray B”;
                break;
        case c:
                cout<<”Hurray C”;
                break;
        default:
                cout<<”Wrong code”;
                break;
  }
}
```

2.2 Iterative Constructs (Looping):

Iterative constructs or Loops enables a program with a cyclic flow of logic. Each statement which is written under the scope of a looping statement gets executed the number of times the iteration/looping

continues for.

In previous chapter we have seen few real life scenarios where a looping construct is needed for performing a particular set of tasks repeatedly for a number of times. Now we will go into details of all looping constructs available in C++.

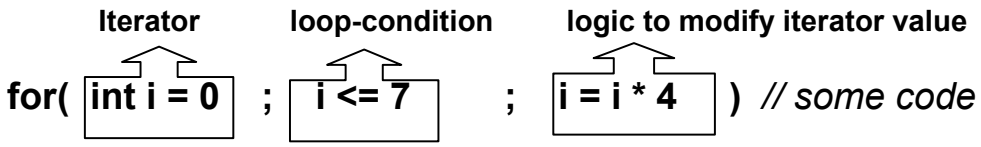
In C++ there are three basic types of looping constructs available, they are :

- while( ) loop // keyword while is used to loop
- do-while( ) loop // keywords do and while are used to loop
- for loop // keyword for is used to loop

It is very important to understand that a looping construct enables repetition of tasks under its scope based on a particular condition called as **loop-condition**. This loop-condition evaluates as true or false. A loop may be continued till a loop-condition is evaluated as true or false, based on a particular program situation.

All the above mentioned three loops have three parts common in them i.e.
- the looping variable (iterator)
- the loop-condition
- logic to change the value of iterator with each cycle/iteration

Let me show you distinguishably these three parts and then we will proceed to the syntax.



Similarly , while and do-while also have these three significant parts as shown below :

```
int i = 0
while ( i<= 7 )
{
    // some code
    i = i * 4 ;
}
```

```
int i = 0;
do
{
        //some code
        i = i *4;
} while(i <= 7 ) ;
```

2.2.1 : while( ) loop construct

A while loop tests for its ending condition before performing its contents - even the first time. So if the ending condition is met when the while loop begins, the lines of instructions it contains will *never* be carried out.

Syntax :

```
while (<loop-condition>)
```

```
{
...;
...;
}
```

A while continues iteration-cycle till its loop condition is evaluated as false. If the loop-condition is false for the first time iteration then loop will not execute even once.

Example :

```
int x = 0, sum = 0 ;
cout<<"Input a natural number";
cin>>x;
while(x > 0 )
{
  sum = sum + x;
  x - - ;
}
cout<<"The sum is :" << sum;
```

The above code snippet finds the sum of n natural number using while loop. The loop is executed till the x is greater than 0 , as soon x becomes 0 the loop is terminated. We observe that within the scope of the loop the value of x is decremented so that it approaches to its next previous value. Thus with each iteration the value of x is added to a variable sum and is decremented by 1.

Lets understand the above program with a Dry Run. Let us assume that user inputs a value 4 for x then at :

1st Iteration start

x is 4 , sum = 0 i.e. the loop-condition 4 > 0 is evaluated as true hence it opens up the while block
==> sum = 0 + 4 = 4
x will be decremented by 1 , thus x = 3

2nd Iteration start
x is 3 , sum = 4 i.e. the loop-condition 3 > 0 is evaluated as true hence it opens up the while block
==> sum = 4 + 3 = 7
x will be decremented by 1 , thus x = 2

3rd Iteration start
x is 2 , sum = 7 i.e. the loop-condition 2 > 0 is evaluated as true hence it opens up the while block
==> sum = 7 + 2 = 9
x will be decremented by 1 , thus x = 1

4th Iteration start
x is 1 , sum = 9 i.e. the loop-condition 1 > 0 is evaluated as true hence it opens up the while block
==> sum = 9 + 1 = 10
x will be decremented by 1 , thus x = 0
5th Iteration start

x is 0 , sum = 10 i.e. the loop-condition 0 > 0 is evaluated as **false** because 0 == 0 hence it locks up the while block and while() loop is terminated.

Thus coming out of while loop block the value of sum is printed as 10
loop executes for 4 times.

Workout yourself :

Consider the program given below :

```
void  main( )
{
    int x = 5, m = 1 , p = 0;
    while( p <= 50 )
     {
          p = x * m ;
          m++;
     }
}
```

Now complete the following Iteration tracking table with details of each iteration, the first one is done for you :

Iteration	loop-condition	x	p	m
1st Iteration	0<= 50 ==> true	5	5	2

Check it out with your teacher.

Why my while loop is not getting executed ?

It is very important to understand here that in while loop the looping condition is evaluated at the beginning of the loop's scope i.e. prior to the entering into scope of loop. This type of checking is called **"Entry Control Checking"**, if the condition fails this checking it will not be allowed into the scope of the loop. That is why while( ) loops are often called as **"Entry Control Loop".** You can imagine a similar checking situation that your tickets are being checked prior to your entry into a cinema hall to view a film. You will not be able to view the film at all if you don't have the tickets !!

Similarly if the loop-condition fails at the first time itself the statements under the scope of loop will not run even once. Observe such a code scenario below :

```
noOfPersons = 4;
noOfTickets = 3 ;
```

```
while( noOfTickets >= noOfPersons )
{
        cout<<"Welcome to Gangs of C++ pure"
}
```

The above loop will not execute at all because the loop-condition is failing at first time.

2.2.2 : do-while( ) loop construct

A do-while loop is identical to a while loop in every sense except that it is *guaranteed* to perform the instructions inside once before testing for the ending condition.

Syntax:

```
do
{
        // do something ;
        // do something ;
} while( <loop-condition> ) ;
```

Example: Consider the following code snippet to find factorial of a given number n :

```
int  f = 1;
int n = 0;
cin>> n;

do
{
        f  = f * n ;
         - - n ;
} while( n > 0 ) ;
cout << "The factorial of  : " << n <<" is " <<f  ;
```

Lets us analyze the execution of the above program : the factorial of any number n is evaluated as

fact = n * (n-1) * (n-2) * ….. * 1

This means that with each iteration the value of n is decremented by 1 and is multiplied with the previous value of n is stored cumulatively. The iteration-cycle stops when n is decremented upto a value equal to 1.

Let us conduct a Dry run on the above code snippet to understand the flow of logic for **n = 4**:

Initially

f =1, n = 4

1st Iteration-cycle:

f = 1 * 4 = 4 , n = 3, condition n > 0 evaluates as true hence loop is continued

2nd Iteration-cycle :

f = 4 * 3 = 12 , n = 2 , condition n > 0 evaluates as true hence loop is continued

3rd Iteration-cycle :

f = 12 * 2 = 24 , n = 1 , condition n > 0 evaluates as true hence loop is continued

4th Iteration-cycle :

f = 24 * 1 = 24 , n = 0 , condition n > 0 evaluates as false , as 0 == 0 hence loop is terminated and program flow comes out of the scope of the loop into program scope.

The output is printed on console as:
"The factorial of : 0 is 24"

Workout yourself :

Though the above factorial code snippet has executed nicely and produced correct and valid result of factorial of 4 i.e. 24 but while prompting output to user it wrongly says that "factorial of :0 is 24" . It must show that "factorial of 4 is 24".

Find out the reason for the above misprinting and correct the program by modifying the code so that it produces expected output prompt to user.

Check your result with your teacher.

Why my do-while() loop is getting executed once even when the condition is not satisfied !

It is very important to understand here that in do-while loop the looping condition is evaluated at the end of the loop's scope i.e. after the last statement in the scope of loop. This type of checking is called **"Exit Control Checking"**, if the condition fails this checking it will not be allowed into the scope of the loop on next iteration-cycle. In the whole affair you are observing that the loop had executed at least once even when condition fails in 1st Iteration-cycle. That is why do-while( ) loops are often called as **"Exit Control Loop".** You can imagine a similar checking situation that your shopping bag items and purchase bill are being checked on your exit out of a shopping mall where you have been for shopping.

For example consider the program 2.3 ahead.

Sentinels :

Often a looping construct executes loop for a fixed number of times based on certain looping condition placed on the looping variable or the iterator, but sometimes the termination of loop is not fixed i.e. the termination of loop depends on some outside input. These types of loops where termination of loop is not fixed and depends on outside input are called **sentinels.**

For example : Consider the program given below :

program 2.3

```
// program to count number of adults and minors

#include<iostream.h>
void main( )
{
        int ad = 0 , m = 0;

        int age = 0;

        cout<< "Input age :";
        cin>> age;
```

```
        while(age > 0)
        {
            if( age >= 18 )
                ad++ ;
            else
                m++;
            cout<< "Input age :";
            cin>> age;
        }

        cout<< "There are total " << ad <<" adults and" << m << "minors";
}
```

Can you tell how many times the loop in above question runs ? Your answer would be probably "No" , or better you will tell that since the termination of the loop depends upon the value of the variable age, it is not fixed that how many times the loop may run.

Yes this is the feature a sentinel has , here the value of variable age controls the execution cycle of the while () loop. If the value of the age inputted by the used before the start of the loop is inputted as 0 then the loop will not be executed at all because the condition fails on the start of the first cycle itself. Moreover it also not fixed that after how many cycles the user may input a 0 value to stop the loop , he/she may input after 3 cycles or 5 or may be 200 !

Workout yourself:

> In the above program 2.3, remove the two shaded input line placed before the start of while () construct, and then execute the code to find whether you are getting same result as before or anything wrong. If incorrect result is produced then find the reason explain it to class friends.

2.2.3 : for(; ;) loop

A for loop has all of its three parts i,e the iterator variable , loop-condition and logic to continue loop kept intact at one place separated by semi colon (;) . This loop is also an **Entry Control Loop,** as condition is checked before entering into the scope of the loop. Let us analyze the syntax of a for looping:

Syntax:

```
for ( <variable(s) initialization> ;  <looping-condition> ; <incrementing / decrementing> )
{
        // do something
        …
}
```

The first part of a for loop is a place where all looping variables are declared and initialized in the same way we declare multiple variables of a single data type : e.g. int i = 0 , j = 2 , k = 9 … ;

This part of the loop is executed only once at the start up of the loop and never it is executed again (how can you re-declare variables with the same name! in same scope). As per turbo c++ compiler the scope of all these variables declared in for loop is same as program-scope i.e. they can be accessed from any other scope in the main( ), but the current ISO C++ has changed this making the scope of all these variable local to for loop only. You will follow the turbo c++ type scoping.

The second part of a for loop defines the looping condition using set of relational and logical operators and governs the number of times the loop would execute. This looping condition is checked before entering into loop, for e.g. i <= j+k etc.

The third part of the for loop defines how to change the value of iterative / looping variables with each cycle. This is very important as to execute the loop for a fixed number of times , for e.g. i++ , j = j+k etc

Let us now integrate all the above three parts to observe how a for loop works :

Example :

```
for(int i = 0 ; i<=3 ; i++)
{
    cout<<"Arun Kumar\n";
    cout<<"Kamal Kant Gupta\n";
    cout<<"Anil Kumar\n";
}
```

The above loop starts with declaring variable i = 0 , then it proceeds to the looping-condition part to check whether condition is satisfied (evaluates as true) so that the flow could enter into the scope of the loop. i.e. 0 <= 3 which is true , the control flow is allowed to enter into the scope of the loop and all the statements (the 3 lines) gets executed one after another. After executing the last statement within the scope of for() the control proceeds to the third part of the loop where it increments the value of looping variable i by 1 , so that the variable is getting the next consecutive higher value i.e. i = 1. Hence the first Iteration of the for( ) loop is finished.

Now at the start of the next iteration the control flow directly proceeds to the second part of the for loop to evaluate looping-condition i.e. 1 <= 3 which is true , the flow of control then proceeds in the similar way as it did in the first iteration and this is repeated till the looping-condition becomes false i.e. when i will become equal to 4 .

Let us put all the things discussed above in the form of an iteration tracking table :

Iteration	i	Looping condition	Output
Starting of loop	0	0<=3 → true	Arun Kumar Kamal Kant Gupta Anil Kumar
End of 1st Iteration	i++ → 1	1<= 3 → true	Same as above
End of 2nd Iteration	i++ → 2	2<= 3 → true	Same as above

End of 3rd Iteration	i++ → 3	3<=3 → true	Same as above
End of 4th Iteration	i++ → 4	4<=3 → false	No entry into for block , loop is terminated

Observe the above table carefully to find that the loop has executed for 4 times, and the last value of the iterative variable on the termination of loop is 4.

Variations in for( ) loop :

All the three parts of a for loop are optional part i.e. they may or may not be present. Observe the valid variation in syntax of for( ) loop :

variation-1 :

```
int i = 0;
for(  ; i<=5 ; i++ )
  cout<<"Hello to all";
```

In the above for( ) loop the first part has been left i.e. we have not declared and initialized the iterator in for( ) , though it has been declared outside the scope of for( ).

variation-2 :

```
int i = 0 ;
for(  ;   ;  i++)
{
  if( i<=5 )
     cout<<"Hello to all";
  else
     break;
}
```

In the above variation-2 the loop is not having first and second part defined for it, though the execution of the program remains same as version 1 since iterator has been declared and the condition for terminating the loop is implemented through a **if-else** construct inside the loop. You may observe that to terminate the loop we have used the **break statement.**
Whenever a break statement is found in a particular construct 's scope it immediately comes out of the current scope.

Variation-3 :

```
int i = 0 ;
for ( ;   ; )
{
    if( i<=5 )
    {
      cout<<"Hello to all";
      i++;
     }
  else
     break;
}
```

The above variation-3 is implemented in a similar to that of the variation-3 but it has its incrementation statement defined within if( ) construct.

In all of the above variations you might have observed that **it is compulsory to put a semicolon (;) within for( ) even when statements before and after it are absent.**

2.3 : Nested Loops

Students in section 2.1.7 we have seen that any conditional construct can be nested within any other conditional construct. Similarly any looping construct can also be nested within any other looping construct and conditional constructs.

Let us look at the following example showing the nesting of a for( ) loop within the scope of another for( ) loop :

```
for(int i = 1 ; i<=2 ; i++)
{
    for( int j = 1 ; j<=3 ; j++)
    {
        cout<< i * j <<endl ;
    }
}
```

Outer for( ) loop

Inner for( ) loop

For each iteration of the outer for loop the inner for loop will iterate fully up to the last value of inner loop iterator. The situation can be understood more clearly as :

1st Outer Iteration
i= 1
1st Inner Iteration
j = 1 , output : 1 * 1 = 1
2nd Inner Iteration
j = 2 , output : 1 * 2 = 2
3rd Inner Iteration
j = 3 , output : 1 * 3 = 3

2nd Outer Iteration
i= 2
1st Inner Iteration
j = 1 , output : 2 * 1 = 1
2nd Inner Iteration
j = 2 , output : 2 * 2 = 4
3rd Inner Iteration
j = 3 , output : 2 * 3 = 6

You can observe that j is iterated from 1 to 2 every time i is iterated once.

Workout Yourself :

Write a program to print a multiplication table from 1 to 10 using concept of nested for loop

Let us summarize what we have learned till now :

i) There are two basic types of Conditional constructs : if() construct and switch-case statement. The if( ) has four different types of variations simple , compound , complex and nested.

ii) There are three basic types of Iterative Constructs : while( ) , do-while( ) , and for( ) Out of these while( ) and for() have similar flow of execution whereas do-while() is bit different.

iii) for( ) and while( ) are called as Entry Control Loop , whereas do-while( ) is called as exit-control loop.

iv) Any of the Iterative construct could be nested within the scope of another iterative construct. A for( ) can be enclosed within scope of another for( ) , while( ) within a for() , a for( ) within a do-while( ) etc.

v) while programming for real life situation we are often going to mix conditional and iterative flow of logic as and when required.

Check your progress:

1. Write C++ program to sum of following series:

i) S = 1 + $\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+...+nterms$

ii) S= 1 + $\frac{1}{1^2}+\frac{1}{2^3}+\frac{1}{3^4}+...+nterms$

iii) S = 1 – 2 + 3 – 5 + 6 – 7 + … n terms

iv) S = 1 + $\frac{1}{x!}+\frac{2^2}{(x-1)!}+\frac{3^3}{(x-2)!}+...\frac{n^n}{(x-n)!}$

2. Using while loop find the sum of digits of number if the inputted number is prime, if number is

then print the nearest prime number.

For example: if user inputs number = 17 then the output would be = 1 + 7 = 8
if user inputs number = 20 then the output would be = 23 (next prime to 20)

3. You might be knowing DNA in Biology , it is a genetic molecule made of four basic types of nucleotides. The four nucleotides are given one letter abbreviations as shorthand for the four bases.

A is for adenine
G is for guanine
C is for cytosine

T is for thymine

When considering the structure of DNA it is made of two strands. Each of these strands are made of long chains of above nucleotides like :

AAGCTCAGAGCTATG 1st strand

TTCGAGTCTCGATAC 2nd strand

Each of the nucleotide of 1st strand pairs with nucleotide in other strand using bond. These bonding obeys the following rule:

I) A will always pair with T and vice versa
II) G will always pair with C and vice versa

as it can be observed in the two strands given above.

Write a program in C++ which allows user to input the nucleotide character sequence of 1st strand and print the probable sequence of the other strand. The user must provided opportunity to input a strand of any size, and only stops when user inputs an out of order character instead of A , G , C , & T.

Chapter -3

Functions in C++

Objectives :

- **to analyze how modularity is implemented in a program at its lowest level.**
- **To appreciate the use and importance of function in C++**
- **to program different types of functions and implement them practically.**
- **To understand difference between User Define Function and Library function.**

3.1: Why to use functions ?

Who likes unnecessary repetition of task ? No body in this world. Every body thinks of re usability these days in every aspect of life. What will you do if your cycle wheel rim breaks down on fine day? Do you will throw the whole cycle or sell the cycle as scrap? No exactly not because cycles are designed in such a way that each of its parts can be repaired or replaced. So you will get a new cycle rim from market and will get it fitted in your cycle! This design of cycle where each of its parts have its own unique functionality and could be reassembled together to form a complete cycle is known as **Modular approach of designing**. Each of the cycles part can thought as a Module which serves some purpose in the whole cycle but is very essential for proper functioning of the cycle. The whole concept is nothing but based on **"Divide and Rule philosophy".** A bigger system is divided into smaller components so that each of these smaller parts could handled easily and effectively. These smaller parts when integrated gives rise to the bigger system.

Just think GOD has also created human beings as a modular entity ! We humans have a body which is integration of organ system and each of these organ system is again integration of some organs. So here organs are acting as modules. These modules (organs) could be taken care of individually when we often fall ill.

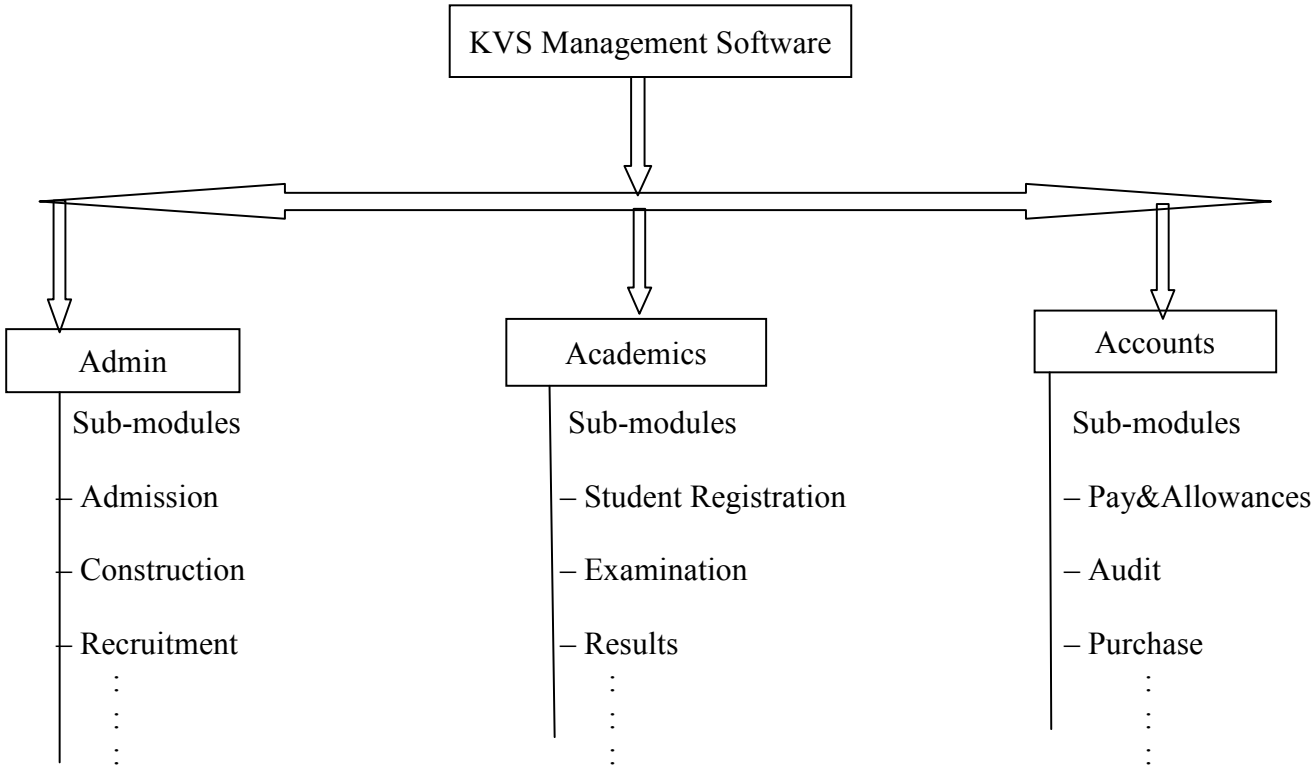
Can you rightly describe what is opposite word for modularity ? Any system which is not modular is known as monolithic (अखंड) or indivisible. A monolithic system does not have any parts or modules, from top to bottom it is one piece.

Software industry has also adopted the modular approach of design where a big software is divided into several modules. Each of the modules are designed for performing specialized task in the whole software. These modules interact with other modules of the system to carry out essential functionality of the whole system.

Each module during its course of execution repeats same type of task, so whenever the whole system requires a specific type of task , for which a particular module is responsible , it calls or invokes that module and the task is done. This calling of module to perform a certain action , can be done several number of times while the software as a whole executes.

Let us understand the above concept with the help of a real life example. Suppose our KVS is going to develop a centralized software for managing all Kvs across the country. While designing such a software KVS has to divide the whole operation of the software into three big modules called as : Admin , Academic and Accounts, each of these modules could be again broken down into many simple and small sub-modules like Admin Module can have Admission , Construction , Recruitment, etc. whereas the Academics can again have sub-modules like Student Registration, Examination , Results etc.

The following diagram describes the whole concept very easily :



When the whole software is divided into modules as seen in the above case scenario then the following benefits could be harvested :

i) Each module can be tracked individually and separately without taking much care of other modules.

ii) A module is a reusable piece of program which could used again and again. Suppose that Navodaya Vidyalaya now wants to make a software like KVS then they can re-use the same modules of KVS with some changes (customization).

iii) If error is found in one module the functionality of that particular module and its associated modules would be disturbed, the whole software will not suffer. Thus errors can be tracked easily and debugged in much less time, because programmer will know which module is causing error, so he will debug that particular module only not the whole software (much like

when you visit doctor suffering from common cold , the doctor does not checks your brain!)

iv) System up gradation (the process of changing from older system to newer one) becomes much easier because only those modules which needs up gradation will be dealt leaving other things as it is.

So we see that modularization of a system gives us much independence and flexibility in terms of fast program development , easier debugging , and re-usability.

How Functions in C++ are related to program modules :

Just as a software is divided into modules , each modules into sub-modules , a sub-module is further divided into several functions. So we may call a **function as a micro-level module** of a bigger software.

A function in C++ :

- is smaller section of code of bigger module/program.
- is re-usable piece of code.
- is very specific in nature as it performs a specific task.
- is often called many times in a program.

Thus a C++ function have all the advantages which a module has in a software.

3.2: Types of function :

Functions in C++ are of two basic types :

a) User Defined : written by a programmer as per his/her requirement domain.
b) Library Function : already available with C++ compiler and stored as Library, from where they can be called and used in any C++ program.

3.2.1 : User Defined Functions :

A user define function in C++ is always created by programmer according to his/her program requirements. Suppose, if some programmer is making software for school management then his list of user defined functions may have functions such as : getFee( ) , calcResult( ) , setExam( ) , all these functions will be used only in school management software not any where else, as they are specially tailored function for school management software.

Let us see the syntax to declare a user defined function :

Function Declaration :

```
<return type> function_name( <parameter list> ) ;
```

where :

return type := is the value which the function returns , if function does not returns any value then we may write there void.

function_name := any valid C++ identifier name

parameter list := declaration of variables of different data types separated by comma these values are inputs passed from outside to the function.

The function declaration as per the syntax given above is also called as **prototype declaration.** In C++ it is compulsory to declare prototype of a function before defining and using it .

The parameter variable in the declaration are also called **Formal parameters.**

Function Definition :

While function definition described about the structure of a function , its inputs and output type , the definition of function actually implements the code of the function. While defining a function we add

C++ code to its block as per requirement.

Syntax :
```
<return type> function_name( <parameter list> )
{   … }
```

Example : Declare and define a function which finds the sum of two integers and returns it.

```
int getSum( int , int );   // declaration of a function

int  getSum( int a  , int b )   // definition of the function
{
   int r = a+b;
   return r;
}
```

The above function declaration has a return type as integer , because the function is meant to return a sum of two numbers. Two numbers to be added are passed to the function as input parameter. The parameter list is having two int separated by a comma (,) it is not compulsory to write a variable names of the parameters in declaration. A semicolon is terminating the declaration of function.

The definition of function is having code written within its scope where the sum is calculated over the passed parameters a and b and the result is returned using a **keyword return.** It is compulsory that the return data type must be same as that of the datatype of the variable returned using return statement.

Workout yourself :

Declare the prototype of a function which :

i) multiplies three integers and return the product
ii) checks whether a passed integer parameter is even or odd
iii) prints your name 20 times.

Consider the few more definition of functions related to various program :

Function *3.1* :

```
        // function to check whether a given number is prime or not
int isPrime(int );
int isPrime(int num )
{
        int count = 0;
        for( int i = 1 ; i <= num ; i++)
          if( num % I ==  0)
             count++;
```

```
        if (count > 2 )
          return 0  ;       // more than two factors means it is not prime , hence a false value is returned
        else
          return 1 ;      // exactly two factors i.e. 1 and num itself means num is prime , hence return a
                          // true value i.e. 1
}
```

In the above function if the passed parameter to the function i.e. num would be a prime it will have exactly two factors counted out in variable count and if not would have more than 2 factors. After we conduct a looping over the num to check its divisibility by every value from 1 to num we get count incremented whenever num is divisible by i (looping-variable). So on the termination of loop the variable count stores the total number of times num gets divisible in the loop.

We check this count value to find whether it is more than two or not, if it is more than two it means num has more than two factors and hence it does not satisfies to be a prime , hence we return an integer 0 to designate that it is not prime , other wise we return 1.

Instead of returning 1 and 0 from function you directly print using cout that num is prime or not, but then don't forget to change the return type of the function **isPrime( ) to void.**

Function 3.2 :

```
// function to print all even numbers found between two given integer parameters

void findSum( int , int);
void findSum(int i , int j )
{
        int sum = 0;
        if( i <= j)
        {
          for( int k = i ; k<= j ; k++)
                if( k % 2 == 0)
                   cout<< k << " ,";
        }
        else
            cout<< "Range not valid";
}
```

Explanation :

In the above function there was no need of returning any value from its scope, we have to print the even numbers found within the for loop. Since a function can return only one value at a time, and once its returns a value the scope of the function finishes all the variables declared in its scope dies off and are no more available to be used again. So instead of returning we printed the multiple outputs using a cout. Look how we have declared the function return type as void when we are not returning any value from its scope.

Function *3.3 :*

```
// program to find the HCF  of two  inputted numbers :
void getHCF( );
```

```
void getHCF( )
{
        int n1 = 0 , n2 = 0;
        int hcf = 0;
        cout<< “Input two numbers whose hcf is to be found”;
        cin>>n1>>n2;
        if( n1 == n2 )
          cout<< n1;
        else if( n1 < n2)
        {
            for( int d = 1  ; d <= n2 ; d++)
            {
                if( n1 % d == 0 && n2 % d == 0)
                {
                        hcf = d;
                }
            }
            cout<< “The hcf is “ << hcf ;
        }
        else
        {

            for( int d = 1  ; d <= n1 ; d++)
            {
                if( n1 % d == 0 && n2 % d == 0)
                {
                        hcf = d;
                }
            }
            cout<< “The hcf is “ << hcf ;
        }
} // end of function
```

Explanation :

Look at the function definition of the above function , since the function is taking the two required inputs within its scope using cout and cin, we had not passed any parameters to function thus we have kept the paranthesis ( ) empty. Also as the function directly prints the output there is no need to return any value from its scope, hence return type is void.

The logic of the function is simple : we are finding the highest of the two numbers and then running a loop from 1 to the highest number and in each iteration we are finding out whether the iterator value divides both the numbers , if it then we put that iterator's value into variable hcf , changing the old value of variable hcf. Thus at the end of the loop the hcf is left out with the Highest Common Factor and we print it with a cout.

Function 3.4 :

```
// function to find the Simple Interest on a given principal , rate , and time where the default value of the
//  time parameter is kept 1 yr.

float   getSimpleInt(float , float , float=1 );
float getSimpleInt(float p , float rate , float time)
```

```
{
        float si  = (p * rate * time)/100;
        return si;
}
```

The above program is having a new type of parameter known as **default parameter.** A C++ function parameter can be made default if that parameter is assigned any constant either at the time of

declaring it or at the time of its definition. When we associate a default value to a parameter as we have associated 1 with last parameter, then this value will be used if the function is getting called without passing this parameter in the call. This we will again see ahead while discussing a function call. The logic of the program is quite easy to understand. Anyway default parameters must be always declared from right to left, otherwise will cause error. i.e you can't make principal alone as default. Following declaration is Invalid.

```
        float SimpleInt(float p = 1000 , float rate , float t=1);
or
        float SimpleInt(float p , float rate = 2 , float t);
```

If you want to make principal as default parameter you have to make all the parameters on its right side default as well.

So the valid declaration in such case would be :

```
float SimpleInt(float p = 1000 , float rate=2 , float t=1);
```

3.2.2 : Calling a C++ Function from a program :

You might be wondering that err!! in the above few function examples , the author has forgot to write a main( ) function (**a main is also a function!) ,** then **how the code written within the scope of the function would be executed?** You might be also wondering that in The general syntax of calling a function would be in above functions 3.1 , 3.2 and 3.4 the **author has not asked the user to input values for variables used in these functions?**

So students it done knowingly , the codes within the scope of a function never gets executed if it not called from the scope of other scope. Just like your badly damaged non-functional land-line is not going to repaired of its own until unless you are not calling a mechanic! . **So a function has to be called from the scope of another function so that the code within the scope of the function works.** Usually we make this call from the scope of main( ) function, **but we can call a function from the scope of any other function if the called function is defined globally.**

The other doubt that author has not provided the input values in few function is also reasonable because a **user should input the parameter values, if not error will be given.** A **user inputs the parameter values from the same scope where from it calls the function.**

Let us know see the syntax of calling a function :

A function call for a non-returning type function (return type is void) :

```
function_name( <parameter_list_if_it is defined_or_leave_it_blank);
```

A function call for a non-returning type function (return type is some **datatype**) :

```
return_datatype <var_name> = function_name( <parameter_list_if_it is
                                                defined_or_leave_it_blank);
```

While calling a function the value of the parameter are also called **actual parameter**.

Now let us call all the four functions which we have defined earlier i.e. Finction 3.1 to 3.4

//calling of function 3.1 : int isPrime(int); from main( )

```
void main ( )
{
   int val = 0;
   cout<<"input a integer to be checked";
   cin>> val;


   int r = isPrime( val );    //second type of calling

   if ( r == 1 )
     cout<< val << " is prime" ;   // function is returning 1 only when it found the num be
                                   // a prime value.
   else
     cout<< val << " is not a prime" ;
}
```

//calling of function 3.2 : void findSum(int , int); from main( )

```
void main ( )
{
   int v1 = 0 , v2 = 0;
   cout<<"input two integer to be checked";
   cin>>v1>>v2;

   int s = findSum( v1 , v2);  //second type of calling

     cout<< "The sum is"<< s ;   // function returns the sum which is caught by s and
                                 // shown!!
}
```

//calling of function 3.3 : void getHCF( ); from main( )

```
void main ( )
{
```

```
        getHCF( );  //first type of calling
                    // though it appears that this type of calling is an easier method, but it is
                    // rarely used in actual software making industries.
    }
```

//calling of function 3.4 : float getSimpleInt(float , float , float=1);; from main( ) ; we may call this //function a different ways let us see

```
    void main()
    {
        float  prnc = 5000, rt = 10 , tm = 5;
        float si = 0.0;
        // first call produces si :  Rs 2500

        si = getSimpleInt(prnc , rt , tm);

        // second call produces si = Rs. 500

        si = getSimpleInt(prnc , rt );  // last parameter is omitted as it was declared as
                                        // as optional type with a default value = 1
    }
```

3.2.3 : Arguments passed by value and by reference

Until now, in all the functions we have seen, the arguments passed to the functions have been passed *by value*. This means that when calling a function with parameters, what we have passed to the function were copies of their values but never the variables themselves. For example, suppose that we called our first function addition using the following code:

```
int x=5, y=3, z;
z = addition ( x , y );
```

What we did in this case was to call to function addition passing the values of x and y, i.e. 5 and 3 respectively, but not the variables x and y themselves.

```
int addition (int a, int b)
                 ↑     ↑
z = addition (  5  ,   3  );
```

This way, when the function addition is called, the value of its local variables a and b become 5 and 3 respectively, but any modification to either a or b within the function addition will not have any effect in the values of x and y outside it, because variables x and y were not themselves passed to the function, but only copies of their values at the moment the function was called.

But there might be some cases where you need to manipulate from inside a function the value of an external variable. For that purpose we can use arguments passed by reference, as in the function duplicate of the following example:

```
// passing parameters by reference
#include <iostream.h>

    void duplicate (int& a, int& b, int& c)
```

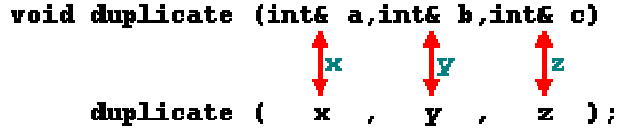
```
{
        a*=2;
        b*=2;
        c*=2;
}

        int main ()
        {
                int x=1, y=3, z=7;
                duplicate (x, y, z);
                cout << "x=" << x << ", y=" << y << ", z=" << z;
                return 0;
        }
```

output : x=2, y=6, z=14

The first thing that should call your attention is that in the declaration of duplicate the type of each parameter was followed by an ampersand sign (&). This ampersand is what specifies that their corresponding arguments are to be passed *by reference* instead of *by value*.

When a variable is passed by reference we are not passing a copy of its value, but we are somehow passing the variable itself to the function and any modification that we do to the local variables will have an effect in their counterpart variables passed as arguments in the call to the function.



To explain it in another way, we associate a, b and c with the arguments passed on the function call (x, y and z) and any change that we do on a within the function will affect the value of x outside it. Any change that we do on b will affect y, and the same with c and z.

That is why our program's output, that shows the values stored in x, y and z after the call to duplicate, shows the values of all the three variables of main doubled.

If when declaring the following function:

***void* duplicate (*int*& a, *int*& b, *int*& c)**

we had declared it this way:

***void* duplicate (*int* a, *int* b, *int* c)**

i.e., without the ampersand signs (&), we would have not passed the variables by reference, but a copy of their values instead, and therefore, the output on screen of our program would have been the values of x, y and z without having been modified.

Passing by reference is also an effective way to allow a function to return more than one value. For example, here is a function that returns the previous and next numbers of the first parameter passed.

// more than one returning value

```
#include <iostream.h>
void prevnext (int x, int& prev, int& next)
{
prev = x-1;
```

```
next = x+1;
}

int main ()
{
  int x=100, y, z;
  prevnext (x, y, z);
  cout << "Previous=" << y << ", Next=" << z;
 return 0;
}
```

Ouput :Previous=99, Next=101

3.2.4 : Arguments passed as const parameter

Look at the following protoype :

```
void myFunction( const int x = 10 );
```

The above function is declaring its first parameter as const i.e even if the user tries to pass value to this argument the new value will not change the value assigned to the const parameter, even in the scope of the function also it will remain unchaged.

Let us see one example :

```
void myFunction( const int x = 10 )
{
      cout<< x;
}

main( )
{
         myFunction(3 ); // even if the function is passed with a value 3 it will not accept.
}
```

The output of the code will be 10 instead of 3.
So whenever you don't want to restrict your parameter to a fixed value , declare it as const. Some compiler may produce error.

check your progress :

1. Write a function (waf) to find the largest out of three integers as input parameters.
2. W.a.f to find whether a number either divisible by its predecessor and successor.
3. W.a.f which returns the sum of digits of any integer passed to it as parameter.
4. W.a.f to calculate discount amount on a given rate of item and discount % offered.

3.1.2 : Library Functions :

These functions are ready-made functions available with C++ compiler. They are stored under various header files. A header file is a normal C++ program file with .h extension containing the code for all the C++ functions defined under it. Header files group functions according to its use and common feature.

Following are some important Header files and useful functions within them :

1.	stdio.h (standard I/O function)	:	gets( ) , puts( )
2.	ctype.h (character type function)	:	isalnum( ) , isalpha( ) isdigit ( ), islower ( ),isupper ( ), tolower ( ), toupper( )
3.	string.h (string related function)	:	strcpy ( ), strcat ( ) strlen( ), strcmp( ) , strcmpi( ) , strrev( ) strupr( ) , strlwr( )
4.	math.h (mathematical function)	:	fabs ( ), pow ( ), sqrt ( ), sin ( ), cos ( ), abs ( )
5.	stdlib.h	:	randomize ( ), random ( ), itoa( ) , atoi().

The above list is just few of the header files and functions available under them , but actually there are many more. If you want to learn their use go to the help menu of your turbo C++ compiler and search out function list and learn its prototype.

The calling of library function is just like User defined function , with just few differences as follows:

i) We don't have to declare and define library function.
ii) We must include the appropriate header files , which the function belongs to, in global area so as these functions could be linked with the program and called.

Library functions also may or may not return values. If it is returning some values then the value should be assigned to appropriate variable with valid datatype.

Let us deal with each of these library functions by calling the them from programs :

gets( ) and puts( ) : these functions are used to input and output strings on the console during program run-time.

gets( ) accept a string input from user to be stored in a character array.
puts() displays a string output to user stored in a character array.

program 3.1 :

```
// program to use gets( ) and puts( )
    #include<iostream.h>
    #include<stdio.h>   // must include this line so that gets( ) , puts( ) could be linked and called
    void main( )
    {
        char myname[25] ;   //declaring a character array of size 25

        cout<<"input your name : ";
        gets(myname)   ;   // just pass the array name into the parameter of the function.
        cout<< "You have inputted your name as : " ;
        puts(myname);
    }
```

i**salnum( ) , isalpha( ), isdigit( ) :** checks whether the character which is passed as parameter to them are alphanumeric or alphabetic or a digit ('0' to '9') . If checking is true functions returns 1.

program 3.2 :

```
// program to use isalnum( ) , isalpha( ), isdigit( )
        #include<iostream.h>
        #include<ctype.h>
        void main( )
        {
                char ch;
                cout<<"Input a character";
                cin>>ch;
                if( isdigit(ch) == 1)
                  cout<<"The inputed character is a digit";
                else if(isalnum(ch) == 1)
                  cout<<"The inputed character is an alphanumeric";
                else if(isalpha(ch) == 1)
                  cout<<"The inputed character is an alphabet.
        }
```

islower ( ),isupper ( ), tolower ( ), toupper( ) : islower( ) checks whether a character has lower case , isupper( ) does opposite. tolower( ) converts any character passed to it in its lower case and the toupper( ) does opposite.

program 3.3:

```
// program to use islower ( ),isupper ( ), tolower ( ), toupper( )
        #include<iostream.h>
        #include<ctype.h>
        void main( )
        {
                char ch;
                cout<<"Input a character";
                cin>>ch;
                if( isupper(ch) == 1)    // checks if character is in upper case converts the character to lowercase
                {
                  tolower(ch);
                  cout<<ch;
                }
                else if(islower(ch) == 1)  // checks if character is in lower case converts the character to
                {                          // uppercase
                  toupper(ch);
                  cout<<ch;
                }
        }
```

fabs ( ), pow ( ), sqrt ( ), sin ( ), cos ( ), abs ( ) :

Program 3.4

```
#include <iostream.h>
#include <math.h>
#define PI 3.14159265 // macro definition PI will always hold 3.14159265

int main ()
```

```
{

 cout<<"The absolute value of 3.1416 is : "<<fabs (3.1416) ;  // abs( ) also acts similarly but only on int data
  cout<<"The absolute value of -10.6 is "<< fabs (-10.6) ;
  cout<<"7.0 ^ 3 = " <<pow (7.0,3);
  cout<<"4.73 ^ 12 = " << pow (4.73,12);
  cout<<"32.01 ^ 1.54 = "<<pow (32.01,1.54);

   double param, result;
   param = 1024.0;
   result = sqrt (param);
   cout<<"sqrt() = "<<result ;

   result = sin (param*PI/180);     // in similar way cos( ) , tan() will be called.
   cout<<"The sine of " <<param<<" degrees is : "<< result ;

  return 0;
}
```

randomize ( ), random ( ), itoa( ) , atoi():

The above functions belongs to header file stdlib.h . Let us observe the use of these functions :

randomize( ) : This function provides the seed value and an algorithm to help random( ) function in generating random numbers. The seed value may be taken from current system's time.

random(<int>) : This function accepts an integer parameter say x and then generates a random value between 0 to x-1

for example : random(7) will generate numbers between 0 to 6.

To generate random numbers between a lower and upper limit we can use following formula :

random(U – L +1) + L

where U and L are the Upper limit and Lower limit values between which we want to find out random values.

For example : If we want to find random numbers between 10 to 100 then we have to write code as :

```
random(100 -10 +1) + 10 ;    // generates random number between 10 to 100
```

Check you progress :

1. Name the header files to which the following functions belongs:

i) toupper( ) , ii) fabs( ) , iii) sqrt( ) , iv) strcpy( )

2. Write a program to check whether a string variable is palidrome or not , using only library function.

----------x----------end of chapter 3----------x----------

Summary :

1. Functions provides modularity to programs.
2. Functions can be re-used.
3. Functions can be of two types : User Defined and Library.
4. We have to declare and define user define functions either in program-scope or header files to use them in a program.
5. To use library functions appropriate header files must be included using #include directive
6. A prototype of a function means , the number , data-types and sequence of its parameters.
7. During function call it is very necessary to pass parameter values (actual parameters) to the function in the same order and type as define in its prototype.
8. The Library functions are kept within header files. These header files are made as per the working nature of the function.
9. A big software must use modular approach of programming to which function is as essential programming element.
10. Sometimes instead of passing values as parameters to the function we pass its references to the function. This type of calling is known as call by reference.

Chapter - 4

Structured Data Types : Arrays and Structures.

Objectives :

- **to understand the meaning of structure datatypes and its availability in C++.**
- **To appreciate the use and importance of Arrays in C++**
- **to differentiate between the use and implementation of different types of Arrays**
- **To use structures as User Defined data type to write programs.**
- **To understand and use typedef**

Structured Data types :

Students till now whatever data type we have used are just primitive data types like int , char , float , etc. All these datatypes are defined within the C++ compiler and that is why they are also called as primitive. We can define a length using int , a weight using float , a name using characters etc. but suppose I tell you “please define a fruit for me in program” ,then your mind starts wondering, as you can't define a fruit just with any one datatype as you did with length , weight etc.

A Fruit itself is a composite entity having following attributes :

color : can be described with a name i.e. char [ ]
taste : can be described with a name i.e. char[ ]
season : can be described with int i.e. 1 for summer , 2 for winter …
price : can be described as float

ans so on...

This means that to describe a fruit we need to have a collection of data-types bundled togeather so that all the attributes of the fruit can be captured. This is true for any real world thing around you say student , mobile , plant etc. So when we bundle many primitive data types together to define a real world thing then it is known as **derived data type or structured data type or User defined data types.**

In this chapter we would look onto two important structured data types , one is **Array** and the other one is **Structure**.

Sometimes we need to have variables in very large quantities , that too of same data type i.e. suppose we want 200 integer variables. In this situation will you declare 200 individual variables ? Absolutely not. This is because it will :

a) wastage of time as well time taking task
b) we won't be able to manage these 200 variables in our program , it will be difficult to remember the names of variable every now and then during programming.

So there exist special structured data type in C++ to tackle this situation. C++ allows a programmer to bundle together all these same type of 200 variable under a same tag name called as Arrays.

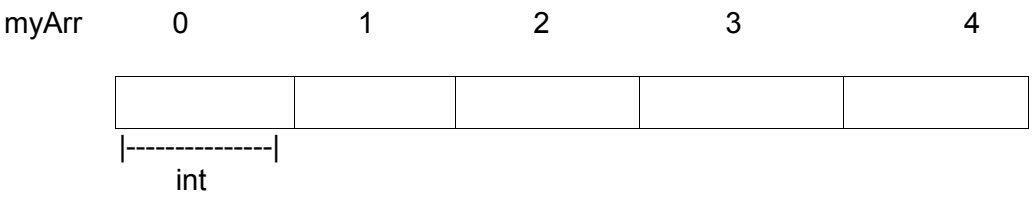
So we are observing that **structuring data type means bundling primitive data type in some or other way so that it solves some special programming situations.**

4.1 Arrays

An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier.

That means that, for example, we can store 5 values of type int in an array without having to declare 5 different variables, each one with a different identifier. Instead of that, using an array we can store 5 different values of the same type, int for example, with a unique identifier.

For example, an array to contain 5 integer values of type int called myArr could be represented like this:



where each blank panel represents an element of the array, that in this case are integer values of type int. These elements are numbered from 0 to 4 since in arrays the first index is always 0, independently of its length.

Like a regular variable, an array must be declared before it is used. A typical declaration for an array in C++ is:

Syntax : `<datatype> array_name [elements];`

where datatype is a valid type (like int, float...), name is a valid identifier and the elements field (which is always enclosed in square brackets [ ]), specifies how many of these elements the array has to contain.

Therefore, in order to declare an array called myArr as the one shown in the above diagram it is as simple as:

***int* myArr [5];**

NOTE: The elements field within brackets [ ] which represents the number of elements the array is going to hold, must be a **constant** value, since arrays are blocks of non-dynamic memory whose size must be determined before execution. In order to create arrays with a variable length dynamic memory is needed, which is explained later in these tutorials.

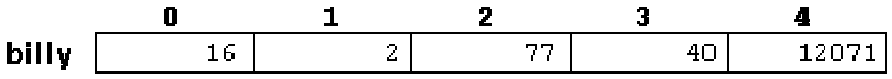
Initializing arrays.

When declaring a regular array of local scope (within a function, for example), if we do not specify otherwise, its elements will not be initialized to any value by default, so their content will be undetermined until we store some value in them. The elements of global and static arrays, on the other hand, are automatically initialized with their default values, which for all fundamental types this means they are filled with zeros.

In both cases, local and global, when we declare an array, we have the possibility to assign initial values to each one of its elements by enclosing the values in braces { }. For example:

***int* myArr [5] = { 16, 2, 77, 40, 12071 };**

This declaration would have created an array like this:



The amount of values between braces { } must not be larger than the number of elements that we declare for the array between square brackets [ ]. For example, in the example of array myArr we have declared that it has 5 elements and in the list of initial values within braces { } we have specified 5 values, one for each element.

When an initialization of values is provided for an array, C++ allows the possibility of leaving the square brackets empty [ ]. In this case, the compiler will assume a size for the array that matches the number of values included between braces { }:

```
int myArr [ ] = { 16, 2, 77, 40, 12071 };
```

After this declaration, array myArr would be 5 ints long, since we have provided 5 initialization values.

Accessing the values of an array.

In any point of a program in which an array is visible, we can access the value of any of its elements individually as if it was a normal variable, thus being able to both read and modify its value. The format is as simple as:

Syntax: array_name[index]

Following the previous examples in which myArr had 5 elements and each of those elements was of type int, the name which we can use to refer to each element is the following:

For example, to store the value 75 in the third element of myArr, we could write the following statement:

```
myArr[2] = 75;
```

and, for example, to pass the value of the third element of myArr to a variable called a, we could write:

```
a = myArr[2];
```

Therefore, the expression myArr[2] is for all purposes like a variable of type int.

Notice that the third element of myArr is specified myArr[2], since the first one is myArr[0], the second one is myArr[1], and therefore, the third one is myArr[2]. By this same reason, its last element is myArr[4]. Therefore, if we write myArr[5], we would be accessing the sixth element of myArr and therefore exceeding the size of the array.

In C++ it is syntactically correct to exceed the valid range of indices for an array. This can create problems, since accessing out-of-range elements do not cause compilation errors but can cause runtime errors. The reason why this is allowed will be seen further ahead when we begin to use pointers.

At this point it is important to be able to clearly distinguish between the two uses that brackets [ ] have related to arrays. They perform two different tasks: one is to specify the size of arrays when they are declared; and the second one is to specify indices for concrete array elements. Do not confuse these two possible uses of brackets [ ] with arrays.

```
int myArr[5];       // declaration of a new array
myArr[2] = 75;      // access to an element of the array.
```

If you read carefully, you will see that a type specifier always precedes a variable or array declaration, while it never precedes an access.

Some other valid operations with arrays:

```
myArr[0] = a;
myArr[a] = 75;
b = myArr [a+2];
myArr[myArr[a]] = myArr[2] + 5;
```

program 4.1

```
// adding all the elements of an array

#include <iostream>
using namespace std;

int myArr [] = {16, 2, 77, 40, 12071};
int n, result=0;

int main ()
{
  for ( n=0 ; n<5 ; n++ )
  {
    result += myArr[n];
  }
cout << result;
return 0;
}
```

Output : 12206

Dynamic Initialization:

Arrays can also be initialized during runtime. The following program shows how to input values into arrays during rum time :

program 4.2

```
// Inputting value in an array during run-time
#include<iostream.h>
main()
{
        int my_arr[5];      // name of array.
        cout<<"\nEnter values at: ";
        for(int  i = 0 ; i < 5; i++)
        {
                cout<<"\n"<<i+1<<" :";
                cin>>my_arr[ i ];                    //stores value at i th index.
        }
}
```

//program 4.3

```
// program to store 10 integers and show them.
#include<iostream.h>
main()
{
        int my_arr[5];      // name of array.
```

```
                cout<<"\nEnter values at: ";
                for(int  i = 0 ; i < 10; i++)
                {
                        cout<<"\n"<<i+1<<" :";
                        cin>>my_arr[ i ];                       //stores value at ith index.
                }
                for(int  i = 0 ; i < 10; i++)
                {
                        cout<<"\Number at "<<i+1<<" :"<<my_arr[ i ];    //show value at  ith index.
                }

        }

//program 4.4
// program to search a particular value out of linear array :
        #include<iostream.h>
        #include<stdio.h>
        #include<process.h>
        main( )
        {
                int a[10] , val = 0;
                cout<<"Input ten integers : ";
                        // inputting array value
                for(int i = 0 ; i<=9 ; i++)
                {
                    cin>> a[i];
                }
                // searching element
                for(int i = 0 ; i<=9 ; i++)
                {
                    if(a[i] == val)
                    {
                        cout<<"Element found at index location : " << i;
                        getch();
                        exit();  // if element is found no need to further search, terminate program
                }
                // if control flow reaches here that means if( ) in the loop was never satisfied
                cout<<"Element not found";
        }



// program 4.5
// program to find the maximum and minimum of an array :
        #include<iostream.h>
        main( )
        {
                int arr[ ] = {10, 6 , -9 , 17 , 34 , 20 , 34 ,-2 ,92 ,22 };
                int max = min = arr[0];
                for(int i = 0 ; i<= 9 ; i++)
                {
                        if(arr[i] < min)
                           min= arr[i];
                        if(arr[i] > max)
                           max = arr[i];
                }
```

```
        cout<<"The minimum of all is : " << min<<"and the maximum is : " <<max;
}
```

Check your progress :

1. Write a program to store 10 elements and increase its value by 5 and show the array.
2. Write the program to divide each of the array element by 3 and show the array.
3. Write a program to find the average of all elements of an array of size 20.
4. Write a program to find second minimum value out of an array containing 10 elements.

4.2 Strings : Array of characters

Strings are in fact sequences of characters, we can represent them also as plain arrays of char elements terminated by a '\0' character.

For example, the following array:

char myStr [20];

is an array that can store up to 20 elements of type char. It can be represented as:

myStr

'H'	'e'	'l'	'l'	'o'	'\0'														
0	1	2	3	4	…			…				…							19

Therefore, in this array, in theory, we can store sequences of characters up to 20 characters long. But we can also store shorter sequences. For example, myStr could store at some point in a program either the sequence "Hello" or the sequence "Merry christmas", since both are shorter than 20 characters.

Therefore, since the array of characters can store shorter sequences than its total length, a special character is used to signal the end of the valid sequence: the null character, whose literal constant can be written as '\0' (backslash, zero).

Our array of 20 elements of type char, called myStr, can be represented storing the characters sequence "Merry Christmas" as:

myStr

'M'	'e'	'r'	'r'	'y'		'C'	'h'	'r'	'i'	's'	't'	'm'	'a'	's'	'\0'				
0	1	2	3	4	…			…				…							19

Notice how after the valid content a null character ('\0') has been included in order to indicate the end of the sequence. The panels in gray color represent char elements with undetermined values.

Initialization of null-terminated character sequences

Because arrays of characters are ordinary arrays they follow all their same rules. For example, if we want to initialize an array of characters with some predetermined sequence of characters we can do it just like any other array:

```
char myword[ ] = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

In this case we would have declared an array of 6 elements of type char initialized with the characters that form the word "Hello" plus a null character '\0' at the end.

But arrays of char elements have an additional method to initialize their values: using string literals.
In the expressions we have used in some examples in previous chapters, constants that represent entire strings of characters have already showed up several times. These are specified enclosing the text to become a string literal between double quotes ("). For example:
"the result is: " is a constant string literal that we have probably used already.

Double quoted strings (") are literal constants whose type is in fact a null-terminated array of characters. So string literals enclosed between double quotes always have a null character ('\0') automatically appended at the end.

Therefore we can initialize the array of char elements called myword with a null-terminated sequence of characters by either one of these two methods:

```
char myword [ ] = { 'H', 'e', 'l', 'l', 'o', '\0' };
char myword [ ] = "Hello";
```

In both cases the array of characters myword is declared with a size of 6 elements of type char: the 5 characters that compose the word "Hello" plus a final null character ('\0') which specifies the end of the sequence and that, in the second case, when using double quotes (") it is appended automatically.

Please notice that we are talking about initializing an array of characters in the moment it is being declared, and not about assigning values to them once they have already been declared. In fact because this type of null-terminated arrays of characters are regular arrays we have the same restrictions that we have with any other array, so we are not able to copy blocks of data with an assignment operation.

Assuming mystext is a char[] variable, expressions within a source code like:

```
mystext = "Hello";
mystext[] = "Hello";
```

would **not** be valid, like neither would be:

```
mystext = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

The reason for this may become more comprehensible once you know a bit more about pointers, since then it will be clarified that an array is in fact a constant pointer pointing to a block of memory.
Using null-terminated sequences of characters

Null-terminated sequences of characters are the natural way of treating strings in C++, so they can be used as such in many procedures. In fact, regular string literals have this type (char[]) and can also be used in most cases.

For example, cin and cout support null-terminated sequences as valid containers for sequences of characters, so they can be used directly to extract strings of characters from cin or to insert them into cout. For example:

```
//program 4.6
//null-terminated sequences of characters

#include <iostream.h>
int main ()
{
  char question[ ] = "Please, enter your first name: ";
  char greeting[ ] = "Hello, ";
  char yourname [80];
  cout << question;
```

```
        cin >> yourname;
        cout << greeting << yourname << "!";
        return 0;
}
```

output : Please, enter your first name: John
Hello, John!

As you can see, we have declared three arrays of char elements. The first two were initialized with string literal constants, while the third one was left uninitialized. In any case, we have to specify the size of the array: in the first two (question and greeting) the size was implicitly defined by the length of the literal constant they were initialized to. While for yourname we have explicitly specified that it has a size of 80 chars.

```
//Program 4. 7
// program to count total number of vowels present in a string  :
#include<iostream.h>
#include<stdio.h>
int main( )
{
        char string[35]; int count = 0;
        cout<<"Input a string";
        gets(string);                   // library function in stdio.h to input a string
        for(int i = 0 ; string[i] != '/0' ; i++)
        {
           if( string[i] == 'a' || string[i] == 'e' || string[i] == 'o' || string[i] == 'u' || string[i] == 'i'
                || string[i] == 'A' ||  string[i] == 'E' || string[i] == 'O' || string[i] == 'U' || string[i] == 'I' )
           {
                count++;
           }
        }
}
```

In the above program the loop is iterated till the character at ith location matches with null character, because after that there is no character left to be scanned. So scanning of characters from first to last character is done using for loop and if vowels are found count keeps incrementing.

String related library functions :

There are few string related library functions in header file string.h which are very useful when we work with strings. They are :

i) **strlen(char[ ]) :** it accepts a string as parameter and returns the length of the string i.e. number of characters within the string. For example strlen("Hurray") will return 6.

ii) **strcpy(char[ ] , char[ ]) :** it accepts two strings as input parameters and then copies the second into the first. For example :

```
char mstr[ ] = "Suresh";
char ystr[20];
strcpy(ystr , mstr);  // copies content of mstr into ystr
puts(mstr);  // prints Suresh
```

iii) **strrev(char[ ]) :** it accepts a string , reverses its content and stores it back into the same string. For example :

```
char myName[ ] = "Kamal";
strrev(myName);
puts(myName) ;    // prints the reversed string as "lamaK"
```

iv) **strcmp(char[ ] , char[ ])** : it accepts two parameters and then compares their content alphabetically, the one which comes first in the acsii chart has considered to be lower. This function returns the value as integer. The integer can be :

0 : if the two strings are equal
+val : if the first string is bigger than the second
-ve : if the second string is bigger than the first.

The above comparison is case sensitive , if we want to perform case insensitive comparison then we have to take another version of the function called as strcmpi( )

Example :

```
if( strcmpi("Kamal" , "kamal") != 0);
    cout<<"Not equal";
else
    cout<<"equal";
```

4.3 Multidimensional arrays

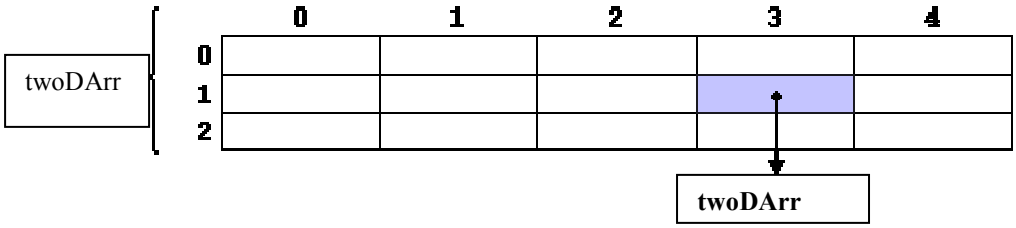
Multidimensional arrays can be described as "arrays of arrays". For example, a two dimensional array can be imagined as a bidimensional table made of elements, all of them of a same uniform data type.

twoDArr represents a bidimensional array of 3 per 5 elements of type int. The way to declare this array in C++ would be:

int twoDArr [3][5];

and, for example, the way to reference the second element vertically and fourth horizontally in an expression would be:

twoDArr[1][3]



(remember that array indices always begin by zero).

Multidimensional arrays are not limited to two indices (i.e., two dimensions). They can contain as many indices as needed. But be careful! The amount of memory needed for an array rapidly increases with each dimension. For example:

char alpha [100][365][24][60][60];

declares an array with a char element for each second in a alpha, that is more than 3 billion chars. So this declaration would consume more than 3 gigabytes of memory!

Program 4.2
// program to show use of a 2D array

```
#define WIDTH 5
#define HEIGHT 3
```

```
int twoDArr [HEIGHT][WIDTH];
int n,m;

int main ()
{
        for (n=0;n<HEIGHT;n++)
        for (m=0;m<WIDTH;m++)
        {
                twoDArr[n][m]=(n+1)*(m+1);
        }
        return 0;
}
```

We have used "defined constants" (#define macro) to simplify possible future modifications of the program. For example, in case that we decided to enlarge the array to a height of 4 instead of 3 it could be done simply by changing the line:

```
        #define HEIGHT 3
```

to:

```
        #define HEIGHT 4
```

with no need to make any other modifications to the program.

Arrays as parameters
At some moment we may need to pass an array to a function as a parameter. In C++ it is not possible to pass a complete block of memory by value as a parameter to a function, but we are allowed to pass its address. In practice this has almost the same effect and it is a much faster and more efficient operation.

In order to accept arrays as parameters the only thing that we have to do when declaring the function is to specify in its parameters the element type of the array, an identifier and a pair of void brackets []. For example, the following function:

```
        void procedure (int arg[])
```

accepts a parameter of type "array of int" called arg. In order to pass to this function an array declared as:

```
        int myarray [40];
```

it would be enough to write a call like this:

```
        procedure (myarray);
```

4.2 Structures In C++

4.2.1 : Why Structures???

To understand the basic need let us proceed with a sample programming scenario.
Problem Definition :

Mr. Chamanlal is Human resource manager in a construction company. He has different responsibilities related to manpower and its utilities. One of his main duties is to keep the record of data related to the Workers who are working in the construction company.

The information which is to be kept regarding a Worker are :

> Name
> Sex
> Age
> Rate per day.

} Data members

Chamanlal wants to write a C++ program which can keep track of the information related to all Workers working in the construction company.

Solution : The above problem could be solved in C++ using 3 ways. They are :

> Using Simple variables to store all information of each employees
> Using Arrays to store related information of each employees
> Using C++ Structures

Let us explore all these three techniques to store information of each Worker of the company. Assuming that there are total 20 Workers.

Using Variables :

As we have learned in class XI that any type of data could be stored in a C++ variable having a suitable data type. For example in the given problem the following data types must be used to store information related to a Worker :

Data	Data Type	C++ Declaration	Total Size in bytes
Name	char	char name[45] ;	45 x 1 = 45 bytes
Sex	char	char sex = 'M'	1 x 1 = 1 byte
Age	int	int age = 15 ;	1 x 2 = 2 bytes
Rate per Day	float	float rate = 100.00 ;	1 x 4 = 4 bytes.

So to store the information of 20 Workers one has to declare a total of :
4 x 20 = 80 variables

The variable declarations in the program would be like :
char name1[45] , name2[45] , name3[45] , … , name20[45] ;

char gender1 , gender2 , gender3 , … , gender4 ;

int age1 , age2 , age 3 , … , age20 ;

float rate1 , rate2 , rate3 , … , rate20 ;

Conclusion :

Using the above methodology , we find that as the number of worker increases the number of variables also increases , and thus it becomes difficult task to manage these huge number of variables.

Using Arrays

The drawbacks of the previous method of information storage could be managed up to some extent , if instead of taking separate variables for storing related information of Workers (say Age) , we could have used an array of Age , which would store the age of all 20 employees.
Though this method does not provides a economic solution in terms of memory , it provides a better management of Memory locations , where the information about the employees would be stored.
So, four different arrays would be required to keep the related information of Workers i,e :

1. An array to keep the Names of the Workers
2. An array to keep the Gender of the Workers
3. An array to keep the Age of the Workers
4. An array to keep the Rate of the Workers.

1.Array to keep Names :

"Ramu"	"Hari"	"kajri"	…	"Bajrangi"
Name[0]	name[1]	name[2]	…	name[19]

Declaration : **char name[20]45] ;**

2. Array to keep Gender:

'M'	'M'	'F'	…	'M'
Gender[0]	gender[1]	gender[2]	…..	gender[19]

Declaration : **char gender[20];**

3. Array to keep Age:

18	22	24	…	19
age[0]	age[1]	age[2]	…	age[19]

Declaration : **int age[20];**
4. Array to keep Rate:

100	120	140	…	100
rate[0]	rate[1]	rate[2]	…	rate[19]

Declaration : **float rate[20];**

Using C++ Structures :

The previous method of storing related data of Workers in different Arrays is a better a solution than storing them in several separate variables. Though it provides better management of information but as the information related to a Worker increases (Say , along with Name , Age , Sex , and Rate , we also want to store Date Of Join , Category , SSN etc.) the need for extra Arrays arises , thus Array management would be another cumbersome issue , if we use Arrays to store information of a Worker.
Thus C++ provides us one of its most fascinating programming construct to handle this situation of storing huge amount of related information about similar entities like WORKER.

To solve the given problem using C++ Structure one has to bundle all the information related to a single WORKER under one single tag name. **This Bundle of information related to an entity,with a Tag Name is called Structure.**

“Ramu”	“Hari”	“kajri”	…	“Bajrangi”
'M'	'M'	'F'	…	'M'
18	22	24	…	19
100	120	140	…	100

Bundled together to form a structure having data related to the worker “RAMU”

Let us now have a look at the syntax for creating a structure :

A Structure in C++ is created by using the Keyword struct. The General Syntax for creating a Structure is :

Syntax :

```
struct < Name of Structure >
{
   < datatype > < data-member 1>;
   < datatype > < data-member 2>;
   < datatype > < data-member 3>;
                …
                …
   < datatype > < data-member n>;
} [ <variable list > ] ;
```

Example :

A Proper example following the previous syntax could be :

```
struct WORKER
{
   char name[45];
   char gender ;
   int age ;
   float rate ;

} W1 , W2 , W3 ;
```

Structure name

data-member

structure variable representing 3 workers.

Points to remember :

1. A structure can declared locally (inside main( )) or globally (Outside main( )).

```
//local declaration
main( )
{
```

```
        struct WORKER
        {
           char name[45];char gender;…
        } W1 , W2 , W3;
           …
           ...
}
```

In this case we can create structure variable only within main( )

```
//Global declaration
struct WORKER
{
   char name[45];char gender;…
};
main( )
{
        WORKER W1 , W2 , W3 ;
…… }
```

In this case we can create structure variable anywhere in the program

If declared globally , the structure variables could be declared both from inside main( ) and any other place outside main( ) function including any other user defined functions.

If declared locally , the structure variables could be declared only within the scope in which the structure has been defined.

2. No data member should be initialized with any value within the structure declaration. i,e the following type of structure declaration is incorrect and cause error :

```
struct WORKER
{
        char name[45];
        gender = 'M';   <=== Invalid Initialization within structure scope
        age = 16;
        rate = 100.00;
};
```

4.2.2 : Structure Variable Initialization

Structure variables could be initialized by two ways :

- Static Initialization (During design time)
- Dynamic Initialization (Passing values during Runtime)

Static Initialization :

```
WORKER w1 ; // structure variable declaration

w1.name = "Ramu";
w1.gender = 'M'; // static initializations
w1.age = 17;
w1.rate = 100.00;
```

The (.) operator here is known as component operator, used to access the data members composing the structure variable.

There is also one another way for static initialization :

```
WORKER w1 = { “Ramu” , ‘M’ , 17 , 100.00 };
```

Warning : The declaration as well as initialization should be in the same line.

Dynamic Initialization :

```
main( )
{
        WORKER w1 ;
        cout<< “Input Worker’s Name”;cin.getline(w1.name , 45);
        cout<< “Input Worker’s Gender”;cin >> w1.gender ;
        …
}
```

4.2.3 Structure variable assignments

We know that every variable in C++ can be assigned any other variable of same data type i,e :

if int a = 7 ; b = 3;

we can write :

a = b ; // assigning the value of b to variable a

Similar is the case with Structure variables also , i,e :

if WORKER w1 = {“Ramu” , ‘M’ , 17 , 100.00};

WORKER w2;

we can write :

w2 = w1 ; // assigning the corresponding individual // data member values of w1 to Worker w2;

or

WORKER w2 = w1;

Note : Both structure variables must be of same type i,e WORKER.

There is a member wise copying of member-wise copying from one structure variable into other variable when we are using assignment operator between them.

So, it is concluded that :

Writing : w1. name = w1.name ;

w1.gender = w2.gender ;

w1.age = w2.age ;

w1.rate = w2.rate ;

is same as :

w1 = w2 ;

Though we can copy the contents of similar types of structure variables , two dissimilar types of structure variables can’t be assigned to each other, even though they may have same types of constituent data members. Assigning values into dissimilar structure variable would cause a incompatible data type error

Consider the two different structures Student , and Worker :

The Student structure can be written as :

```
struct Student
```

```
{
  char name[45];
  char gender;
  int age ;
  float height;
};
```

Compared to Worker Structure , we find that , the data types and sequence of all the data members of structure student is same as that of structure Worker, but still we can never write :

Worker w = {“Ramu” , ‘M’ , 20 , 5.5};

So the following assignments are invalid assignments :

```
Student s = w ;    // Invalid assignment as s and w both are two different structures
```

or

```
Student s ;
s = w;              //Invalid
```

4.2.4 Array of Structure :

Just like we can make array of integers , floats , chars we can also make array of user defined data types like structures. The syntax to make such an array is :

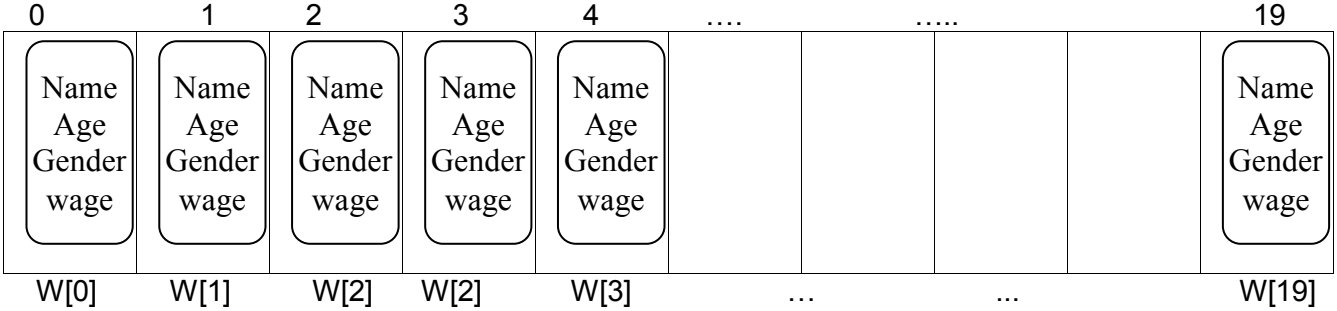
<structure_name> <array_name>[size];

where : structure_name is the name of the structure which you have created.
array_name is any valid identifier
size is a positive integer constant.

Example : to create array of 20 Workers we can have :

Worker W[20];

The above structure could be visualized as :



Each of the elements of the array is itself a structure hence each of them have all the four components.

Initialization of Structure Array :
Array of structures like any other array can be initialized statically and dynamically (runtime).

Static initialization : The above array of Workers W[ ] can be statically initialized as :

```
W[ ] = {
            { “Ramu”, 'M' , 17 , 100.00  },
            { “Hari” , 'M' , 22 , 120.00 },
            { “kajri” , 'F' , 18 , 100.00 },

              …          //values of other 16 workers.
              …
            {“Bajrangi” , 'M' , 24  , 140.00 }
      };
```

Note : The data-member values must be supplied in the same order of the data-member described in structure definition , otherwise datatype mismatch error may cause.

Dynamic Initialization :

```
#include<iostream.h>
#include<stdio.h>
main( )
{
      Worker  W[20] ;       // consider the Worker structure declared earlier.

      for(int i = 0 ; i<19 ; i++)
      {
        cout<<”Input the name, age , gender and wages of worker number”<<i+1;
        gets(W[ i ].name) ;  // inputting the name of worker at index i
        cin>>W[i].age ; // inputting the age of worker at index i
        cin>>W[ i ].gender ;
        cin>>W[i].wage;
      }

}
```

4.2.5 : Function with Structure variable:

Just like we can pass various primitive datatypes to a function we can also pass a structure variable as function parameter / arguments. The benefit is that the structure carries a bundled information to the structure. The prototype of such a function would be :

```
<return_type> function_name(<structure_name> <var> , … , … );
```

Let us understand the concept with following program :

```
//program  4.8
// program to write a function which increases the wage of a female worker by passed percent

      void increaseWage(   Worker  & w ,  float incr )
      {
            if( w.gender == 'F' || w.gender == 'f' )
            {
```

```
                w.wage + = w.wage* (incr /100) ;
        }
}
```

Look at the highlighted parameter, we have passed formal structure variable as reference, so that the increment is reflected back in actual parameter.

Similarly , if we don't want to pass our structure variable as reference we can do so , but then we have to return the modified structure variable back. This can be achieved in above function, if we take return type as structure name. Let us modify the function so that it returns a structure :

```
Worker increaseWage( Worker w , float incr)
{
        if( w.gender == 'F' || w.gender == 'f' )
        {
                w.wage + = w.wage* (incr /100) ;
        }
  return w ;
}
```

4.2.6 : Use of typedef statement :

A typedef can be used to indicate how a variable represents something. This means that we are defining another name for an existing datatype. Let us see the following example to understand this.

Suppose we are going to write a program in C++ where we need lot of strings of size 30 say. This means that every time we have to declare character array of size 30 , like char arr[30] or so. Instead of declaring each time with size specification if we have a declaration like :

```
string arr ;
```

which automatically instructs the compiler that arr will be a character array of size 30 , then this will give a good look to our program. We can achieve this by using typedef statement as done in the program given below :

```
#include<iostream.h>
typedef char  string[30] ;
main( )
{
        string name ;
        cout<<"Input your name";
        gets(name);
        cout<<"You name has  "<< strlen(name)<<"ciharacters";
}
```

The second line defines a new datatype named as string which is nothing but array of 30 characters.

Check your Progress :

Q1. consider the structure distance given below :

```
struct Distance
{
        int feets , inches; };
```

Now answer the following :

i) what does feets and inches known as ?

ii) declare a variable of type Distance and initialize it statically.

iii) Can we use = operator between two Distance type variables. If yes, what does it infers?

iv) write a function which takes two Distance variable as parameters and then returns their sum. For example say D1 = {9 , 6 } and D2 = {5 , 8 } , then D1 + D2 = 15 feet 2 inches

v) Declare an array representing 10 Distances , and then write a program to initialize all these 10 distances and then increase all the distances by 5 feets.

Q2. Consider the structure called Point as :

```
struct Point
{
        int xCoord;
        int yCoord;
};
```

Write a function which accepts three Points as parameter and then checks whether the three points are collinear.

Q3. Consider the following structure called Element and Compound :

```
struct Element
{
        int atomic_no;
        float atomic_mass;
        char symbol[2];
        int valency;
};
struct Compound
{
        Element Elements[ 5 ];  // array to keep all elements present in the compound.
        float molecular_mass;
};
```

Write a program to initialize a compound and show its details.

Summary :

1. Structured data types are used in special programming situations like arrays are used when there is bulk of similar type of variables to deal with, whereas structures models a real life entity having some attribute.

2. Arrays can have both primitive as well user defined data types. There are basically two types of arrays , one dimensional and multidimensional (2 dimensional)

3. Functions can be passed structured datatypes as its parameter as well as they can return them.

4. typedef is a special keyword to define a another name for a variable definition.

Sample Paper Class – I
Subject – Computer Science

Time: 3Hours

Maximum Marks: 70

Note. (i) All questions are compulsory.

1 a) What are the different functions of operating system? 2
b) How the information can be used as a data explain ? 2
c)What do you mean by unary operators 2
d) What are the different parts of CPU? Explain every part in brief. 2
e) Define System Software and what are its two main types? Give examples.2
f) What is Booting? 1
g) Which of the following are hardware and software? 1
(i) Capacitor (ii) Internet Explorer (iii) Hard disk (iv) UNIX

2. Explain the following term: (Give answer any six) 6

i) Variable
ii) Token
iii) Array
iv) Debugging
v) Comment
vi) Keyword

3 a) What is the difference b/w **"while"** & "do **while**" loop? 2
b) What are data types? What are all predefined data types in c++? 2
c) What will be the size of following constants? 1
'\v','\v",
d) Write the corresponding C++ expressions for the following mathematical expressions: 1
i) $\sqrt{(a^2+b^2)}$ (ii) $(a+b)/(p+q)^2$
e) Evaluate the following, where p, q are integers and r, f are floating point numbers.
The value of p=8, q=4 and r=2.5
(i) f = p * q + p/q
(ii) r = p+q + p % q 2

4 a) What is the output of the following? 2

i)
```
# include<iostream.h>
 void main ( )
        {
         int i=0;

                cout<<i++<<" "<<i++<<" "<<i++<<endl;
                 cout<<++i<<" "<<++i<<" "<<++i<<endl
        }
```

ii)
```
# include<iostream.h>
 void main( )
        {
        a=3;
        a=a+1;
        if (a>5)
                 cout<<a;
         else
                cout<<(a+5);
```

```
            }                                                                    2
```
iii) What will be the output of the following program segment? 3

If input is as: (a) g (b) b (c) e (d) p

```
cin >>ch;
switch (ch)
 { case 'g': cout<<"Good";
   case 'b': cout<<"Bad";
             break;
   case 'e': cout<<" excellent ";
             break;
   default: cout<<" wrong choice";
 }
```

iv) Determine the output: 2

```
for(i=20;i<=100;i+=10)
   {
    j=i/2;
    cout<<j<<"";
   }
```

v) What output will be the following code fragment produce?

```
void main( )
          {
          int val, res, n=1000;
          cin>>val;
          res = n+val >1750 ? 400:200;
  cout<<res;
           }
```

(i) if val=2000 (ii) if val=1000(iii) if val=500 3

5 a) Find the error from the following code segment and rewrite the corrected code underlining the correction made. 2

```
# include(iostream.h)
void main ( )
 int X,Y;
 cin>>>X;
 for(Y=0,Y<10, Y++)
   if X= =Y
          cout<<Y+X;
  else
         cout>>Y;      }
```

b) Convert the following code segment into switch case construct. 3

```
int ch;
  cin>>ch;
  If(ch = = 1)
  {     cout<<" Laptop";
  }
    else If(ch = = 2)
        {
        cout<<"Desktop ";
        }   else if(ch= = 3)
```

```
            {
              cout<<“Notebook”;
            } else
                    {
            cout<<“Invalid Choice”;
                      }
          }
    }
```

c) Convert the following code segment into do-while loop. 3

```
#include<iostream.h>
void main()
     {   int i;
      for(i=1;i<=20;++i)
          cout<<”\n”<<i;
     }
```

d) Given the following code fragment

```
int ch=5;
cout << ++ch<< “\n”<<ch<<”\n”;
```

i) What output does the above code fragment produce?
ii) What is the effect of replacing ++ ch with ch+1? 2

6 a) Which header files are required for the following?
(i) frexp()((ii) sqrt( ) (iii) rand( ) (iv) isupper() 2

b) Evaluate: 4
i) $(12)_{10} = (X)_2$
ii) $(347)_8 = (X)_{10}$
iii) $(896)_{16} = (X)_8$
iv) $(100)_{10} = (X)_2$

7 a) Write a C++ program to check a year for leap year or not. 2
b) Write a C++ program to check a number for Armstrong or not. 4
c) Write a C++ program to calculate the factorial of any given number 4
d) Write a C++ program to print the Fibonacci series 4
e) Write a C++ program to print table a given number. 2

Answer key

Q.No.1

a. Major OS functions are listed below
1. Process Management 2. Storage Management 3. Information Management (Student has to describe all in brief)

b. The processed information can be used as a data again to produce a next level information. For example- totoal no. of students school wise can give the information that how students are there in one region again this information as a data ca be used to calculate that how many students are studying in KVS

c. unary operators are the operators, having one operand and two operators. There are two types of unary operators-
i. unary increment (Ex. a++(post increment))/++a(pre increment))
ii. Unary decrement (a—(post decrement)/--a(pre decrement))

d. ALU(Arithmetic logic unit), CU(control unit), MU(memory unit)

e. System software are the software that govern the operation of computer system and make the hardware run. These software can be classified into two categories. Operating System & Language Processor

f. Booting is a process through which operating system makes the computer system ready to perform user's task

g. Hardware- I&III, Software- II&IV

Q.No.2

i. variable is a name given to the memory location, whose value can be changed during run time.

ii. The smallest individual unit in a program is known as a token

iii. Array is a combination of similar data values. It is used to store more than one value under same name

iv. debugging is a way to correct the errors in the program during compilation

v. comments are non executable statements, used to give the information about the code for future use.

vi. Keywords are the reserved words, programed by the programmer to perform the specific task. The keyword can not be taken as a name of variable.

Q.No.3

a. While loop is entry control loop i.e. while loop first will test the condition and if condition is true then only the body of the loop will be executed. While do-while loop is exit control loop and even the condition is not true at least one time the body of the loop will be executed.

b. data types are means to identify the types of data and associated operation of handling it. The fundamental data types are- char, int , float , double and void .

c. one byte

d. i. sqrt(a*a+b*b) & ii. ((a+b)/((p+q)*(p+q))

e. students do yourself

Q.No.4

a. i. 0 1 2
4 5 6 , ii. 9 , iii. For g- good & bad/ for b – bad / for e – excellent / for – p wrong choice
iv. 10,15,20,25,30,35,40,45,50 v. 400, 400, 200

Q.No.5

a. Errors – if x==y (correct- if(x==y)) & cout>>y(correct cout<<y)

b.

```
int ch; cin>>ch;
 switch(ch)
       {
          Case 1 :  cout<<" Laptop"; break;
           Case 2:  cout<<"Desktop "; break;
           Case 3:  cout<<"Notebook";break;
           Default : cout<<"Invalid Choice";
```

```
        }
c. #include<iostream.h>
        void main()
            {   int i;
             i=1
              do
                { cout<<"\n"<<i;
                   ++i
                 }while (i<=20);
             }
```
d. In both condition output will be 6 5

Q.No.6

a. math.h , math.h , stdlib.h , ctype.h

b. 1100, (232) , (4226), (1100100)

Q.No.7

a.

```
#include<iostream.h>
#include<conio.h>

void main()
{
 clrscr();
 int year;
 cout<<"Enter Year(ex:1900):";
 cin>>year;
 if(year%100==0)
 {
  if(year%400==0)
   cout<<"\nLeap Year";
 }
  else
   if(year%4==0)
    cout<<"\nLeap Year";
   else
    cout<<"\nNot a Leap Year";
 getch();
}
```

b.

```
#include<iostream.h>
#include<conio.h>
void main()
{
int Number,Temp,b=0;
cout<<endl<<"Enter any number to check";
cin>>Number;
Temp=Number;
int P;
while(Temp>0)
{
 P=Temp%10;
 b=b P*P*P;
 Temp=Temp/10;
}
```

```
if(b==Number)
{
 Cout<<endl<<"Armstrong no";
}
else
{
 cout<<"Not an armstrong no";
}
getch();
}
```

c.

```
#include <iostream.h>
int factorial(int);

void main(void) {
int number;

cout << "Please enter a positive integer: ";
cin >> number;
if (number < 1)
cout << "That is not a positive integer.\n";
else
cout << number << " factorial is: " << factorial(number) << endl;
}

int factorial(int number) {

if(number <= 1) return 1;
else
return number * factorial(number - 1);
}
```

d.

```
#include<iostream.h>
#include<conio.h>
int main()
{
        clrscr();
        unsigned long first,second,third,n;
        int i;
        first=0;
        second=1;
        cout<<"how many elements(>5)? \n";
        cin>>n;
        cout<<"fibonacci series\n";
        cout<<first<<" "<<second;
        for(i=2;i<n;i++)
        {
                third=first+second;
                cout<<" "<<third;
                first=second;
                Second=third;
        }
        return 0;
        getch();}
```

e.

```
#include<iostream.h>
#include<stdio.h>
void main()
{
int r,m,i,n;
cout<<"Enter the number to generate its table";
cin>>n;
cout<<"Enter the number(table upto)";
cin>>m;
i=1;
while(i<=m)
{
r=n*i;
cout<<N<<"*"<<I<<"="<<R<<endl;
}

}
```

Sample Paper - II
Subject – Computer Science

Max Marks 70 Duration 3 hrs

Note:- All questions are compulsory

Q1)

a) What is significance of My Computer ? 2

b) Explain different types of operating systems . 1

c) What is an Operating System? Explain its any two functions 2

d) How is a compiler different from interpreter? 2

e) Convert

(i) $(10.10)_{10} = (\ ?\ )_2$

(ii) $(101011.1110)_2 = (\ ?\ )_{10}$ 2

Q2)

a) What do you mean by run time error? 1

b) what is the deference between if and if – else statement 2

c) Mention and explain briefly any three characteristics of a good program 2

d) How can you give a single line and multiline comments in C++ explain with suitable examples 2

e) what do you mean by header files? What is the difference between #include <iostream.h> and #include"iostream.h" 2

Q3)

a) Classify the following variable names of c++ into valid and invalid category 2

(i) 1no (ii) num 1 (iii) num (iv) num1num (v) num+1 (vi) num.1

b) Give output of following code. 3

```
#include<iostream.h>
int m=5; void check();
void main( )
{  int m=20;
        {
        int m=10*::m;
        cout<<"m="<<m<<"::m="<<::m<<endl;
        } check();
        cout<<"m="<<m<<"::m="<<::m<<endl;
        check(); cout<<"::m="<<::m<<"m="<<m<<endl;
        }
          void check()
          {  ++m;

    }
```

c) What will be result of following statements if a=5 , b=5 initially 2
(i) ++a<=5 (ii) b++<=5

d)Name the header file(s) that shall be needed for successful compilation of the following C++ code. 2

```
void main( )
{
  char name[40];
  strcpy(name,"India");
  puts(name); }
```

e) Explain conditional operator (?) with example in c++ . 2

f) What is difference between '/' and '%' operators in c++ ? explain with a example 2

Q4)

a) What do you mean by function prototype in C++ 2

b) Explain Break and Continue statement in C++ with example. 2

c) Find syntax error(s) if any in following program (Assume all header files are present) 2

```
 main<>
{  int c;
  switch( c );
case 1.5:  { cout<<" India is great\n";
            }  break;
'case' 2:  {  cout<<" hello\n";
            }  break;
}  // end of main
} // end of switch
```

d) How will you declare and define 1
i)Array named mark with 10 integer values
ii)array named avg with 8 float values

e) Convert following while loop to for loop 2

```
int   x=0;
while(x<=100)
{  cout<<" the value of x is \n"<<x;
   cout<<"done \n";
  x+=2;
}
```

f) Define token ? 1

g) What is the difference between call by value and call by reference explain with suitable example 2

h)Find the output of the following program; **3**

```
#include<iostream.h>
#include<ctype.h>
void main( )
{ char Text[ ] = "Comp@uter!";
for(int I=0; Text[I]!='\0';I++)
{ if(!isalpha(Text[I]))
Text[I]='*';
else if(isupper(Text[I]))
Text[I]=Text[I]+1;
else
Text[I] = Text[I+1]; }
cout<<Text; }
```

i) What are differences between for and do- while loop ? explain with example 2

j) How can you define Global Variable and Local Variable? Also, give a suitable C++ code to illustrate both. 2

Q5 : a) Write a program to print the left and right diagonal element of an NXN matrix 4

- **b)** b. Write a program to find the factorial of a number recursive function. 4
- **c)** Write a program to find the total number of characters, lines and words in a paragraph of text. 4

d) Write a program to sort an array on N numbers in ascending order. Avoid duplication of elements. 4

e) Write a program to find the roots of a quadratic equation. 4

Answer Key

Q.No.1

a. My Computer is used to viewing the contents of a single folder or drive. Also the things we have on our computer – Your Programs, Documents and data files, for example- are all accessible from form one place called My Computer.

b. Types of operating system -
i. Single User ii. Multiuser iii. Time Sharing iv. Real Time v. Multiprocessing vi. Interactive

c. An operating system is a program which acts as an interface between a user and the Hardware.
Functions- i. Process Management and Storage(Memory) Management.
Note. Students has to explain both in brief

d. An interpreter converts source program(HLL) into object program(LLL) line by line whereas compiler converts source program into object program in one go and once the program is error free it can be executed later .

Q.No.2

a. An error in logic or arithmetic that must be detected at run time is called run time error. For example any number is divided by zero (c=a/0) is run time error

b. if statement can perform only true condition and there is no option for false condition while if-else statement can perform both true as well as false condition . Example,

```
if(a<b) cout<<" a is greater than b" – in this code for false condition no statement is given
if(a<b) cout<<" a is greater than b"
 else
        cout<<" b is greater than a" – here for both option either true or false output will come
```

c.

d. Comments in c++ can be given in two way
Single line comment – which can be given by using the symbol - //
Example- // Program to find the sum of two numbers
And Multi line comment – which can be given by using the symbol /*…….*/
Example -- /* this is the way
To give multiline Comment */

e. Header files are the files in which some pre – programed program will be stored by the developer of the language which will help the user of to develop their program. If the header file is written in angular bracket(<>) compiler will search it in C++ library only while in case of " " the header file will be searched throughout the database.

Q.No.3

a. i , ii ,v , vi – are invalid & iii, iv are valid declaration

b. m=5 ::m=5
m=20 ::m=6
m=7 ::m=20

c. True and false

d. String.h and stdio.h

e. The conditional operator (**? :**) is a ternary operator (it takes three operands). The conditional operator works as follows:

- The first operand is implicitly converted to **bool**. It is evaluated and all side effects are completed before continuing.
- If the first operand evaluates to **true** (1), the second operand is evaluated.
- If the first operand evaluates to **false** (0), the third operand is evaluated.

The result of the conditional operator is the result of whichever operand is evaluated — the second or the third. Only one of the last two operands is evaluated in a conditional expression
Ex.- // expre_Expressions_with_the_Conditional_Operator.cpp

```
// compile with: /EHsc
// Demonstrate conditional operator
#include <iostream>
using namespace std;
int main() {
   int i = 1, j = 2;
   cout << ( i > j ? i : j ) << " is greater." << endl;
}
```

f. Operator / is used to find the quotient while % is used to find the remainder in the c++ expression for ex.
int a=5,b=2,c; if c=a/b the result of c will be 2 and if c=a%b; the value of c will be 1

Q.No.4
a. A function prototype is a declaration of the function that tells the program about the type of the value returned by the function and the number and the type of arguments.
The prototype declaration looks just like a function definition except that it has no body.
Ex. void abc();
b. break makes the compiler to transfer the control out of the loop...

```
for
{....
if
{
true/false condition
break;
}
}
here;
```

the control is transferred to "here"

continue makes the compiler to execute the next iteration of the loop

```
for
{
Condition..
continue;
****;
}
```

**** will not b executed and next iteration will happen
continue can be given inside an if condition

e. to j . Do yourself.

Ans 5 :
a)

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
void main( )
{
        int A[10][10];
        int i,j,N;
        clrscr( );
        cout<<"\nHow many rows and columns required for matrix: ";
```

```
                cin>>N;
                cout<<"\nEnter "<<N*N<<" elements: ";
                for(i=0;i<N;i++)
                {
                        cout<<"Enter the elements into Row "<<i+1<<": ";
                        for(j=0;j<N;j++)
                           cin>>A[i][j];
                }
                clrscr( );
        }

        cout<<"\nThe entered elements in the matrix are: \n";
        for(i=0;i<N;i++)
        {
                for(j=0;j<N;j++)
                cout<<A[i][j]<<"\t";
                cout<<endl;
        }
        cout<<"\n\n\nThe elements which are belongs to only diagonals...\n";
        for(i=0;i<N;i++)
        {
           for(j=0;j<N;j++)
             if((i==j)||((i+j)==(N-1)))
                cout<<setw(6)<<A[i][j];
            else
                cout<<" ";
                cout<<endl;
        }
        getch( );
}
```

b**)**

```
#include<iostream.h>
#include<conio.h>
long f =1;
long factorial(int n)
{
  if (n==0)
    return f;
else
    f=n*factorial(n-1);
}
void main( )
{
    clrscr( );
    long num;
    cout<<"\nEnter the number to which you want to find factorial: ";
    cin>>num;
    cout<<"\nThe factorial of the number = "<<factorial(num);
    getch( );
}
```

c)

```
#include<iostream.h>
```

```
#include<conio.h>
#include<stdio.h>
void main( )
{
  char str[300];
  int i,charcount=0,words=1,lines=1;
  clrscr();
  cout<<"\nEnter the Paragraph ie message: \n";
  gets(str);
  for(i=0;str[i]!='\0';i++)
   {
        charcount++;
        if(str[i]==' ')
          words++;
       if (charcount%80==0)
          lines++;
   }
 cout<<"\nNumber of Characters in the entered message: "<<charcount;
 cout<<"\nNumber of Words in the entered message: "<<words;
 cout<<"\nNumber of Lines in the entered message: "<<lines;
 getch( );
}
```

d)

```
#include<iostream.h>
#include<conio.h>
void main( )
{
          clrscr( );
          int A[20],N,i,j,temp;
          cout<<"\nEnter the number of elements:";
          cin>>N;
          for(i=0;i<N;i++)
          cin>>A[i];
          //Bubble sort technique
          for(i=0;i<N;++i)
             for(j=0;j<(N-1)-i ;j++)
               if(A[j]>A[j+1])
                  {
                     Temp=A[j];
                      A[j]=A[j+1];
                      A[j+1]=Temp;
                      }
cout<<"The Elements in the array after sorting…. ";
for(i=0;i<N;i++){
cout<<A[i]<<'\t';
}
```

e)

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main( )
```

```
{
  clrscr( );
  double d1,d2,b,a,c,d;
  cout<<"\nEnter the value of b,a and c: ";
  cin>>b>>a>>c;
 d=(b*b-sqrt(4*a*c));
 if(d==0)
          cout<<"\nRoots are equal or distinct";
 else if(d>=0)
          cout<<"\nRoots are Real";
 else
 {
     cout<<"\nRoots are complex..ie Imaginary";
     d1=(-b+d)/(2*a);
     d2=(b+d)/(2*a);
     cout<<"\nD1: "<<d1;
     cout<<"\nD2: "<<d2;
     getch( );
  }
}
```

Sample Paper - III
Subject – Computer Science

Time: 3Hours

Maximum Marks: 70

Note. (i) All questions are compulsory.

Q.No.1

a. Define Analog an Digital Computer
b. why analytical engine often called the pioneer computer.
c. what are the different types of printer. Explain in brief
d. What is the difference between copying and moving files and folders
e. What functions are performed by an operating system as a resource manager

Q.No.2

a. Define the term ASCII and ISCII.
b. What are the different Data representation schemes
c. Explain different types of RAM
d. What do you mean by ports. Explain in brief all types of ports 4
e. Why primary memory is termed as '**destructive write'** memory but 'non-destructive read' memory
f. Explain the different features of OOP. 3

Q.No.3

a) Find the output of the following program. 3

```
#include<iostream.h>
        void Withdef(int HisNum=30)
        {
          for(int I=20;I<=HisNum;I+=5)
          cout<<I<<",";
          cout<<endl;
        }
        void Control(int &MyNum)
        {
           MyNum+=10;
           Withdef(MyNum);
        }
        void main()
        {
          int YourNum=20;
          Control(YourNum);
          Withdef();
          cout<<"Number="<<YourNum<<endl;
        }
```

b.
```
void main()
  {
        char *NAME="a ProFiLe!";
        for(int x=0;x<strlen(NAME);x++)
        if(islower(NAME[x]))
          NAME[x]=toupper(NAME[x]);
        else
```

```
        if(isupper(NAME[x]))
          if(x%2!=0)
            NAME[x]=tolower(NAME[x-1]);
        else
            NAME[x]--;
            cout<<NAME<<endl;
    }
```

c. How Many time the following code will be executed

```
        int i = 1 ;
        i= i - 1     ;
        while(i)
        {
                cout<<"it's a while loop";
                i++   ;
        }
```

d. What is difference between Actual parameter and Formal parameter? Give an example in C++ to illustrate both type of parameters.

e. Explain different jump statements with suitable example.

f. rewrite the following code using do-while loop

```
        int i ;
        for(i=0;i<10;i++)
        {
           cout<<i; i++ cout<<i;
        }
```

g. identify the error(s) in the following code fragment

```
        char ch; int v=0,o=0;
        cout<<"enter character";
        while ((ch>='A' && ch<='Z')||( ch>='a' && ch<='z))
        {
          switch(ch) {
                case 'a':
                case 'e':
                case 'i':
                case 'o':
                case 'u':
                case 'U': ++v; break; default : ++o;
        }cout<<v;<<" "<<o;
        }
```

Sample Paper - IV
Subject – Computer Science

Time: 3Hours
Maximum Marks: 70

Note. (i) All questions are compulsory.

Q.No.1.

a.Differentiate between primary memory and secondary memory.. Give examples of each type of memory. 2

A)The memory inside the CPU is primary memory (main memory) and the memory outside it is known as secondary(auxiliary) memory.
Primary Memory: RAM (Random Access Memory) and ROM (Read only Memory) comes under primary memory. RAM is volatile memory and ROM is non volatile memory. All the data and programs must be stored in RAM for execution. But the content of RAM is not permanent.
Eg: RAM, ROM.
Secondary Memory: Since primary memory has a limited storage capacity and is not permanent, secondary storage devices are used to store large amount of data permanently. There are various types of secondary devices available these days.
Eg: Hard disks, Floppy disks ----- Magnetic Media
CD ROMS, DVDs ----- Optical Media

b. What is the difference between semantics error and syntax error? Give an example of each. 3

A) 1.Syntax Errors: Syntax errors occur when rules of a programming language are misused.
Syntax refers to formal rules governing the construction of valid statements in a language.
Eg: int a,b //Did not keep ; (semicolon) at the end of statement.
2.Semantics Error: Semantic errors occur when statements are not meaningful.
Semantics refers to the set of rules which give the meaning of a statement.
Eg: X * Y = Z;
(Siva plays Guitar is Syntactically and Semantically correct but
Guitar plays Siva is Syntactically correct but Semantically incorrect).

c. What do you meant by a lexical unit? Give an example. 2

A) Token: The smallest individual unit in a program is known as a token (**Lexical Unit**).
There are 5 types of tokens.
(i) Keywords (ii) Identifiers (iii) Literals (iv) Punctuators (v) Operators
d. what is relation between Microprocessor and Microcomputer 2
Ans. Microcomputer is a computer that contain a Microprocessor
e. Briefly Distinguish between a general purpose and special purpose computer 2

Q.No.2

a. what is the shortcut menu ? what is its significance 2
b. what is the significance of Recycle Bin 2
c. Explain the concept of time sharing 2
d. What is meant by the term multiprogramming and multitasking 2.
e .Explain briefly the function performed by an operating system as processor manager. 2

Q.No.3

a. **4.**Differentiate between object oriented programming and procedural oriented programming with the help of examples of each. 3

A. **Procedure Oriented Programming:**

A program in a procedural language is a list of instructions where each statement tells the computer to do something. The focus is on the processing. There is no much security for the data.

Object Oriented Programming:

The object oriented approach views a problem in terms of objects involved rather than procedure for doing it. In object oriented programming, object represents an entity that can store data and has its interface through functions

b.Distinguish between if and switch statement. 3

A)The if-else and switch both are selection statements and they both let you select an alternative out of given many alternatives by testing an expression.
But there are some differences in their operations.
(i) The switch statement differs from the if statement in that switch can only test for equality whereas if can evaluate a relational or logical expression. Ie multiple conditions.
(ii) The switch statement selects its branches by testing the value of same variable whereas the if else construction lets you use a series of expressions that may involve unrelated variables and complex expressions.
(iii) The if-else can handle ranges whereas switch cannot.

c.Differentiate between call by value and call by reference with help of an example. 4

A)(i)In call by value, actual arguments will be copied into the formal perameters.
In call by reference, formal perameters are references to the actual arguments.
(ii)In call by value, if any modification is occurred to the formal perameter, that change will not reflect back to the actual argument.
In call by reference, if any modification is occurred to the formal perameter (reference to the actual argument), the actual argument value will be changed.
(iii) We should go for call by value when we don't want to modify the original value.
We should go for call by value when we want to modify the original value.
(iv) Example:

```
void Change(int a, int &b)
{ a= 2*a;
b=20;
}
```

Here a is called by "call by value" method and b is called by "call by reference"
So as the value of a is changed, actual argument for a will not be changed,
as the value of b is changed, actual argument for b will be changed

d.Write the names of the header files of the following functions.
i)getch() ii) isalpha() iii)strcpy() iv)sqrt() 2

A) (i) getch( ) - conio.h (ii)isalpha( ) – ctype.h
(iii)strcpy( ) - string.h (iv)sqrt( ) - math.h

Q.No.4
a. What will be the output of the following program? 3

```
int main()
{
  int i=0,x=0;
  for(i=1;i<10;i*=2)
  {
    x++;
    cout<<x;
  }
  cout<<"\n"<<x;
}
```

b.Rewrite the following program after removing the syntactical error(s) if any.
Underline each correction. 3

```
#include<iostream.h>
int main
{
  struct movie
   {
     char movie_name[20];
     char movie_type;
     int ticket_cost=100;
   }movie;

  gets(movie_name);
  gets(movie_type);
}
```

A)

```
#include<iostream.h>
#include<stdio.h>
void main( )
{
  struct movie
 {
   char movie_name[20];
   char movie_type;
   int ticket_cost;
 }Movie;
Movie.ticket_cost=100;
gets(Movie.movie_name);
cin>>Movie.movie_type;
}
```

c. **Define the terms Polymorphism & Inheritance.** **2**

d.Write the output of the following program. 3

```
#include <iostream.h>
void main()
{
    int x = 5;
    if(x++ = = 5)
      cout<<"five"<<endl;
    else
```

```
        if(++x = = 6)
          cout<<"Six"<<endl;
}
```

e)
What will be the output of the following segment? 4

```
struct number
{
   int no1, no2;
};

void display(number n)
{
   cout<<"Number1="<<n.no1++<<"Number2="<<- -n.no2<<endl;
}
void main( )
{
   number n1={10,100}, n2, n3;
   n3 = n1;
   n1.no1 + = 5;
   n2 = n3;
   n2.no1 - = 5;
   n2.no2 * = 2;
   n3.no1 + = 1;
   display(n1);
   display(n2);
   display(n3);
}
```

f. convert $(4A8c)_{16}$ into binary number 2

Q.No.5

a. Write a program to find the factorial of any given number 3
b. Assuming suitable data types give necessary declaration for an array of 20 voter records each record of which consists of four data values viz. id_no,name, address, age, make use of above declaration to write program segment that print id_no and name for all those whose age exceeds 60. 5
c. Write a program to add two matrices of same order MXN. 4
d. Write a program to find either given no is prime or not 3
e. Write a function to find simple interest 3

Sample Paper -V
Subject – Computer Science

Time: 3Hours

Maximum Marks: 70

Note. (i) All questions are compulsory.

Q.No.1

a. Write the header file for the given function 2
abs(), isdigit(), sqrt(), setw()

b. Define Microcomputer 1

c. what is data and what is the output of data processing system 2

d. what is the function of memory and what are its measuring units 2

e. what do you mean by language processor. 2

f. what is the difference between save and save as command. 2

g. Expand the following: 2
i)CPU ii) ROM iii)MICR iv)CD-R 2

Q.No.2

a. Rewrite the following code after removing the syntax error if any. Underline the correction 2

```
#include <iostream.h>
jampto(int, int )
void main()
{
        first=10, second=20;
        jumpto(first;second);
        jumpto(second);
}
void jumpto(int n1, intn2=20)
{
   n1=n1+n2;
}
```

(b) Find the output : 3

```
void result(int &x, int y=10)
{
   int temp = x + y;
   x + = temp;
   if(y <=10)
     y + = temp;
}
void main( )
{
   int A1=10, B2=5;
   result(A1, B2);
   cout<<A1<<B2<<endl;
   result(A1);
   cout<<A1<<B2<<endl;
   result(B2);
   cout<<A1<<B2<<endl;
}
```

D. Find the output 3

```
 #include <iostream.h>
void main( )
{
   int i = 0, x = 0;
  do
  {
    if(i % 5 = = 0)
    { cout<<x;
     x++;
    }
   ++ i;
  }while(i<10);
  cout<<"\n"<<x;
}
```

e. Find the correct possible output 3

```
    void main()
     {
       randomize();
       char city[][10]={"del","chn","kol","bom","bng"};
       int fly ;
       for(int i=0;i<3;i++)
       {
          fly = random(2)+1;
          cout<<city[fly]<<":"
       }
     }
```

Output:-

1. Del:chn;kol
2. Chn:kol:chn
3. Kol:bom:bng
4. Kol:chn:kol

Q.No.3

a. what do you mean by literals? What are the different kind of literals 3
b. Explain the types of errors in c++ 2
c. what is the difference between structure and arrays 2
d. what do you mean by dynamic and static allocation 3
e. what is the difference between 'a' and "a" in c++. 1
f. What do you mean by code generation 2

Q.No.4

a. What do you mean by nested structure? Explain with suitable example 2
b. What is data abstraction? Explain the concept with the help of an example.
c. Convert the following equations to C++ statements. 3

i) s = 1+1/x+1/x2+1/x3

A) s = 1 + 1/x + 1/(x*x) + 1/(x*x*x);

ii) V = 4/3Πr3

A) V = 4/(3*3.1415*r*r*r);

d. Explain any two string handling functions with syntax and examples 3

e. i)$(AC.20)_{16}$= $(?)_2$= $(?)_8$ ii) **(**$4A56)_{16}$= $(?)_{10}$= $(?)_8$
f. Find the 1's and 2's complement of 128. 3
g. explain a nested for with suitable example 2

Q.No.4

a. Write a program to print the diagonal (left & right) elements of an N□N matrix. 4

A)
//Program to print the left and right diagonal element of an NXN matrix

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
void main( )
{
   int A[10][10];
   int i,j,N;
   clrscr( );
   cout<<"\nHow many rows and columns required for matrix: ";
   cin>>N;
   cout<<"\nEnter "<<N*N<<" elements: ";
   for(i=0;i<N;i++)
   {
       cout<<"Enter the elements into Row "<<i+1<<": ";
       for(j=0;j<N;j++)
        cin>>A[i][j];
   }
  clrscr( );
  cout<<"\nThe entered elements in the matrix are: \n";
  for(i=0;i<N;i++)
   {
       for(j=0;j<N;j++)
         cout<<A[i][j]<<"\t";
         cout<<endl;
   }
  cout<<"\n\n\nThe elements which are belongs to only diagonals...\n";
  for(i=0;i<N;i++)
 {
   for(j=0;j<N;j++)
   if((i==j)||((i+j)==(N-1)))
    cout<<setw(6)<<A[i][j];
   else
    cout<<" ";
    cout<<endl;
 }
getch( );
}
```

b. Write a program to find the factorial of a number recursive function. 4

A)

```
#include<iostream.h>
#include<conio.h>
```

```
long f =1;
long factorial(int n)
{
   if (n==0)
     return f;
   else
   f=n*factorial(n-1);
}
void main( )
{
   clrscr( );
   long num;
   cout<<"\nEnter the number to which you want to find factorial: ";
   cin>>num;
   cout<<"\nThe factorial of the number = "<<factorial(num);
   getch( );
}
```

c. **3.**Write a program to find the total number of characters, lines and words in a paragraph of text. 4

A)

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
void main( )
{
  char str[300];
  int i,charcount=0,words=1,lines=1;
  clrscr();
  cout<<"\nEnter the Paragraph ie message: \n";
  gets(str);
  for(i=0;str[i]!='\0';i++)
   {
      charcount++;
      if(str[i]==' ‘)
        words++;
      if (charcount%80==0)
        lines++;
   }
  cout<<"\nNumber of Characters in the entered message: "<<charcount;
  cout<<"\nNumber of Words in the entered message: "<<words;
  cout<<"\nNumber of Lines in the entered message: "<<lines;
 getch( );
}
```

d.Write a program to sort an array on N numbers in ascending order. Avoid duplication of elements. 4

A)

```
#include<iostream.h>
#include<conio.h>
void main( )
{
  clrscr( );
```

```
  int A[20],N,i,j,temp;
  cout<<"\nEnter the number of elements:";
  cin>>N;
  for(i=0;i<N;i++)
    cin>>A[i];
  //Bubble sort technique
   for(i=0;i<N;++i)
     for(j=0;j<(N-1)-i ;j++)
        if(A[j]>A[j+1])
          { Temp=A[j];
            A[j]=A[j+1];
            A[j+1]=Temp;
          }
 cout<<"The Elements in the array after sorting…. ";
 for(i=0;i<N;i++)
 cout<<A[i]<<'\t';
}
```

e..Write a program to find the roots of a quadratic equation. 2

A) #include<iostream.h>

```
#include<conio.h>
#include<math.h>
void main( )
{
  clrscr( );
  double d1,d2,b,a,c,d;
  cout<<"\nEnter the value of b,a and c: ";
  cin>>b>>a>>c;
  d=(b*b-sqrt(4*a*c));
  if(d==0)
    cout<<"\nRoots are equal or distinct";
 else if(d>=0)
   cout<<"\nRoots are Real";
 else
  cout<<"\nRoots are complex..ie Imaginary";
  d1=(-b+d)/(2*a);
  d2=(b+d)/(2*a);
  cout<<"\nD1: "<<d1;
  cout<<"\nD2: "<<d2;
  getch( );
}
```